NORCE Norwegian Research Centre AS
www.norceresearch.no

Drilling Data Hub

Drilling Data Semantics

Benoît Daireaux, Nejm Saadallah

NORCE Energy, REPORT 1 – 2020

Prosjekttittel:	Demo2000 Drilling Data Hub
Prosjektnummer:	100258
Institusjon:	NORCE Energy
Oppdragsgivere:	NFR, Total, AkerBP, Vår Energi, Equinor

Gradering:	Open
Rapportnr.:	1-2020
ISBN:	978-82-8408-084-0
Antall sider:	103
Bildetekst og kreditering:	NORCE

Oslo, 11.03.2020

Benoit Daireaux	Fionn Iversen	Helga Gjeraldstveit
Project leader	*QA*	*Leader*

Disclaimer

NORCE assumes no responsibility or liability for any errors or omissions in the content of this research report. The information contained in this report is provided "as is", based on their best knowledge and effort during the work of the project with no guarantees of completeness, and accuracy.

1. Table of Contents

Figures

2. Introduction

Deploying advanced automation solutions on a large scale is a current challenge. Experience shows that a considerable amount of work must be performed before every single use of those solutions: we refer here to the necessity of correctly identifying and understanding the various real-time signals available at the rig site.

This configuration work is often regarded in the industry as a onetime process: signals should be identified only once, namely during the installation phase of the considered system. The underlying assumption is that all real-time signals can be associated to a particular physical device, typically a sensor, and that the sensory system of the rig is not modified very often, such that adjustments in the system's configuration occur at a low frequency. In practice, this "seldom updates" assumption is erroneous. Even if the drilling machinery itself (e.g. draw-works, top-drive, mud pumps) is not updated very frequently (although maintained), the sensory configuration of the drilling process is different from drill-string run to drill-string run, and changes during the course of a drilling operation. The archetypical example is the downhole sensory system, that depends on the precise construction of the BHA, which typically changes from run to run. Surface hydraulic configuration is also a dynamic component of the rig: pits can be aligned, trip tank can be used, density measurements can be made at different locations, under different conditions. Further, mounting of a top-drive, in place of the elevator, introduces variations in the way hook-load measurements are made or should be interpreted.

Even when restricting a review of drilling real-time signals to measurements, one rapidly concludes that the configuration of those signals is far from being static. Consequently, suppliers of automation solutions need to provide a continuous service purely devoted to the correct management of real-time data. This service is costly and introduces risk of erroneous configuration, as much of the work done here is manual.

It is the goal of the Drilling Data Hub (DDHub) to provide tools to reduce such workload and risk to a minimum: the considered solution to reach this goal is also seen as a potential enabler for advanced automation solutions, paving the way for automatic interoperability between the actors on the drill-floor and thereby full autonomy.

3. Drilling automation, real-time data and interoperability

The present work has been initiated to facilitate the design, development, deployment and adoption of drilling automation systems. As such, it is part of a larger process. One may consider the entire well life cycle:

1. Planning
2. Drilling
3. Completion
4. Production
5. P&A

Each stage of this process is connected to the previous one, and as a consequence also to the following stages. In particular, the physical state of the wellbore evolves continuously, during each of the last four stages, and predictions are constantly made for the wellbore's shape for the near and far future. The way the wellbore has evolved in the past, its present state as well as its probable future are all important information, that should in principle be carried from one stage of the process to the next in an automated fashion.

The work done in this project focuses on the second stage of the process, namely drilling. With respect to the overall picture it will provide insights into how to represent the most relevant information, namely the drilling data, and as such contribute to the overall integration process.

As opposed to the sequential view of the well's life-cycle, we consider the concurrent operations that can take place during the second stage, drilling. Indeed, several tasks are simultaneously performed by different actors for the duration of the well construction process. Machine control, geo-steering, gain/loss monitoring, fluid mixing, and drilling optimization, to name a few, are integral parts of the process and are all interrelated activities. The work presented in this report has to be seen in view of this "vertical" integration (we refer to Figure 1 for the signification of the horizontal and vertical axes).

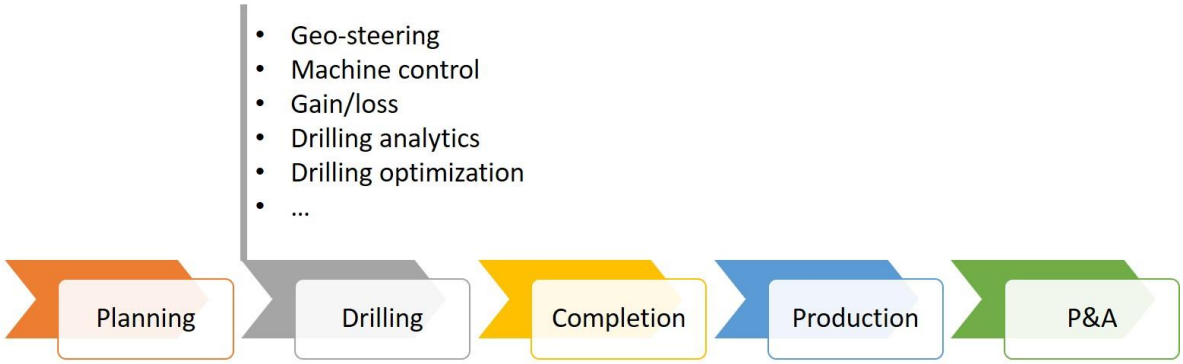

Figure 1: Life-cycle of a drilling well, with some of the concurrent operations that take place during the drilling phase.

All of the mentioned activities involve as per today some amount of manual work: even the most automated parts of the process still require manual configuration, monitoring, and maintenance. A general objective of the drilling industry is to remove -or at least reduce as much as possible – the needs for manual work, and automate as much as possible. The goal of the Drilling Data Hub is to support this objective through providing tools and mechanisms for automated seamless integration of the various drilling automation systems.

Drilling automation

The range of application of automation solutions is becoming wider and wider. Digitalized processes, semi-automated workflows or machine automation are present at all the steps of the well construction process, from planning to completion. Several classifications exist: in (Macpherson J. D., et al., 2013) one considers Monitoring, Advice, Control and Autonomy as the main families of automation (see Figure 2), while (Annaiyappa, 2013) defines three categories: Machine Automation, Drilling Process Automation and Drilling Advisory Automation.

Figure 2: a classification of drilling automation systems, from (Macpherson J. D., et al., 2013).

Below is a (non-exhaustive) list of typical drilling automation applications:

- Monitoring systems: several applications ((Mayani, Rommetveit, Oedegaard, & Svendsen, 2018), (Chmela, Abrahmsen, & Haugen, 2014)) exist that continuously model the drilling process, based on inputs from the real operations. Such simulations, sometime called digital twins, serve as the basis for drilling events detection, which may be either short term events (pack-off, pipe wash-out) or long term events (cuttings accumulation).
- Vibration reduction surface control: vibration (especially stick-slip) can be to a certain extent eliminated by fine control of the top-drive torque and rotation speed. The two main existing systems are called Soft-torque (Kyllingstad & Nessjøen, 2009) and Z-Torque (Dwars, Lien, Øydna, & Baumgartner, 2019), (Dwars, 2015).
- Geo-steering: relying on automatic interpretation of LWD data, novel solutions are emerging that allow for automatic geo-steering (Luo, et al., 2015) (Maus, et al., 2020). The integration with active control of the directional component is often still lacking, so these solutions are often used in advisory mode, which is the normal first testing phase for new technologies.
- Drilling optimization: automatic selection of drilling parameters to ensure high ROP together with low drilling risk is often called drilling optimization. Several systems exist (Dunlop, et al., 2011) (Spivey, et al., 2017) (Payette, et al., 2019) that perform such operations. More are currently being developed.
- Drilling process automation: as per today, the most advanced automation systems deal with automation of the process itself. This implies estimation of operational safeguards (safe combinations of hook velocity, top-drive speed or mud pump rates with respect to downhole conditions), dynamic sensor limits and optimum parameters for predefined drilling sequences such as friction test or pump start-up. Such system are now commercially available, and can be run on both fixed installations (Cayeux, Daireaux, & Dvergsnes, 2011) (Cayeux, Daireaux, & Dvergsnes, 2011) or floating ones (Daireaux, Dvergsnes, Cayeux, Bergerud, & Kjøsnes, 2019).
- Drilling autonomy systems: the grail of drilling automation research is to devise autonomous systems, where all decisions occurring during the process are taken by computer systems, without any human assistance. Even if such systems don’t exist as per today, prototypes are currently being designed and developed (Cayeux, Mihai, Carlsen, & Stokka, 2020), that draw a path for the future of drilling automation.
- Mud mixing: with the recent developments in automated mud properties measurements (Taugbøl, Brevik, & Rudshaug, 2019), the automation of the mud mixing process is emerging (see for example (Nafikov & Glomstad, 2013), where density control is often the core concern.

Real-time data

Common to all the applications listed above is the central role played by real-time drilling data. None of these applications can work without access to this source of information. Note that other types of information can be mandatory: wellbore configuration, control protocol specifications, drilling procedures and drilling plan can be required to properly set up a given automation system, but even the most frugal of these need real-time drilling data.

Real-time drilling data has always been used: purely manual operations still involve human monitoring of the main sensor readings (hook-load, surface torque, stand-pipe pressure, pit volumes) without which the proper handling of the drilling machinery is not possible. However, the value of real-time data gets significantly more important when considering automation tasks. In (Macpherson J. , de Wardt, Laing, & Zenero, 2016) the authors characterize this added value for different parts of the well construction process in view of manual operations, monitoring and control applications (see Figure 3). It is clearly shown that control applications (and more generally speaking active drilling automation) are the type of applications that benefit the most from real-time data. This emphasizes the importance of having a reliable and transparent way of accessing the different data generated during the drilling process.

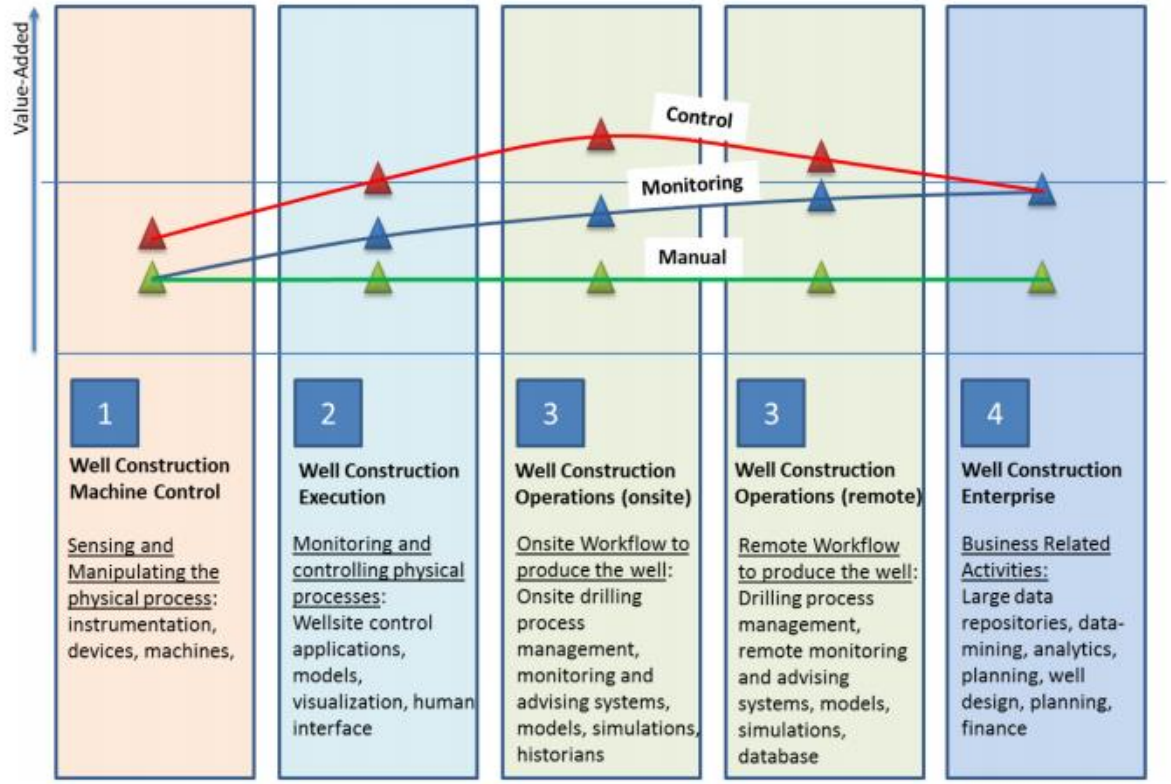

Figure 3: added value of real-time data, from (Macpherson J. , de Wardt, Laing, & Zenero, 2016)

Several recent publications discuss alternatives for the distribution of real-time signals. In (Shields & Brackel, 2014), a classification of the different data exchange standard distinguishes between the purely IT related (TCP/IP for basic networking, HTTP, REST, SOAP for WebServices, XML or JSON for data formatting or standard message buses such as JMS, WCF, AMQP), the industry-wide standards (all the fieldbus ones such as Modbus, CanBus or Profibus, OPC-UA or DDS) and the standards specific to the oil industry (Wits, WitsML, LIS, LAS, DLIS, PPDM). From this analysis, it is clear that proper and efficient communication at the rig site, between the rig site and external location and up to cloud solutions is technically feasible, and the authors suggest using OPC-UA as an aggregation layer for connection between real-time analysis (such as control applications) and integrated analysis (such as Remote Operation Center analyses).

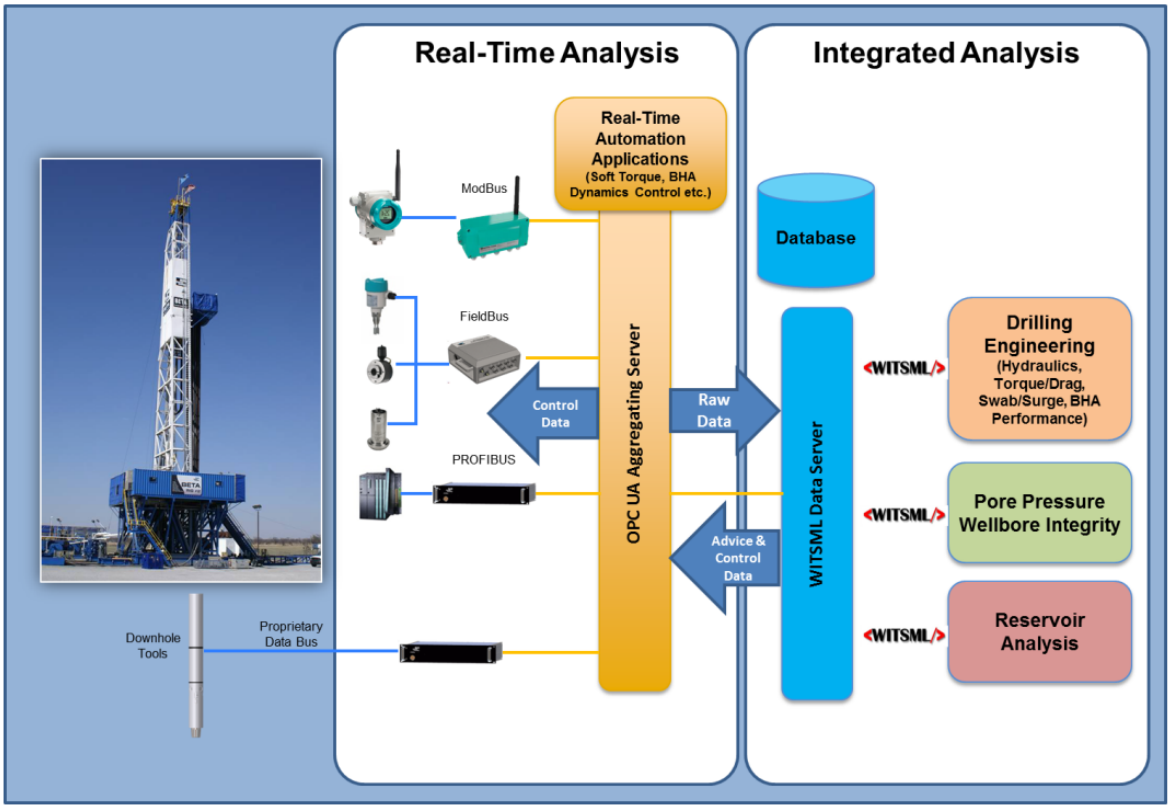

Figure 4: in (Shields & Brackel, 2014), the authors suggest the above architecture for aggregation and external distribution of drilling data.

Cloud architectures and their benefits with regards to automation, monitoring and process advisory are discussed in (Annaiyappa, 2013),. As a final view on the subject, the analysis of timely requirements for various applications, coupled with available communication techniques (see (Macpherson J. D., et al., 2013) and Figure 5) clearly excludes some solutions (such as WitsML for its high latency) for control application, but illustrate the fact

that a common data description language may be beneficial at all the levels of their automation pyramid (Figure 5).

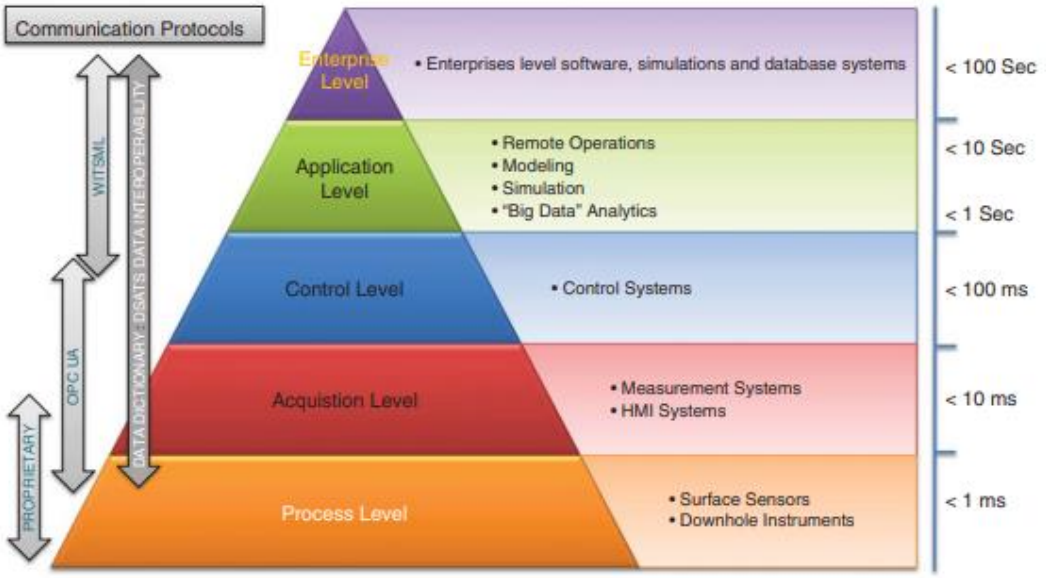

Figure 5: Purdue Automation Pyramid, from (Macpherson J. D., et al., 2013).

The very nature of the drilling data that is to be considered is more seldom discussed. It is acknowledged in the DSA-Roadmap report (DSA-R, 2019) that real-time data encompasses surface and downhole sensor data and commands - derived or unstructured. However, it should also be recognized that relevant real-time data may also cover simulations, calculations, or dynamic machinery limits, and is not limited to sensor readings.

Interoperability

There exist several definitions of interoperability. Wikipedia[1] defines it as follows:

"Interoperability is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, at present or in the future, in either implementation or access, without any restrictions."

In the same source, the useful distinction between syntactic and semantic interoperability is made:

[1] https://en.wikipedia.org/wiki/Interoperability

"If two or more systems use common data formats and communication protocols and are capable of communicating with each other, they exhibit syntactic interoperability. Beyond the ability of two or more computer systems to exchange information, semantic interoperability is the ability to automatically interpret the information exchanged meaningfully and accurately in order to produce useful results as defined by the end users of both systems."

The DDHub project's ambition is to enable semantic interoperability as defined above.

Although there exist a huge variety of protocols involved during drilling operations, it does not currently affect the ability of drilling automation applications to operate properly. Most of the industrial actors that develop such applications have implemented drivers and communication interfaces towards all automation systems. Even if syntactic interoperability is not yet achieved, most of the building blocks are already in place. This contrasts strongly with the semantical interoperability, which is almost nonexistent. In (Sadlier & Laing, 2011), a detailed analysis of the current state of the drilling industry in terms of interoperability is performed, and an emphasis is made on the benefits interoperability entails in terms of innovation. Sadlier and Laing refer in particular to a classification of interoperability (originally taken from (Gasser & Palfrey, 2008), see Figure 6) where for each of the interoperability steps, clear gains in terms of productivity and innovation are facilitated.

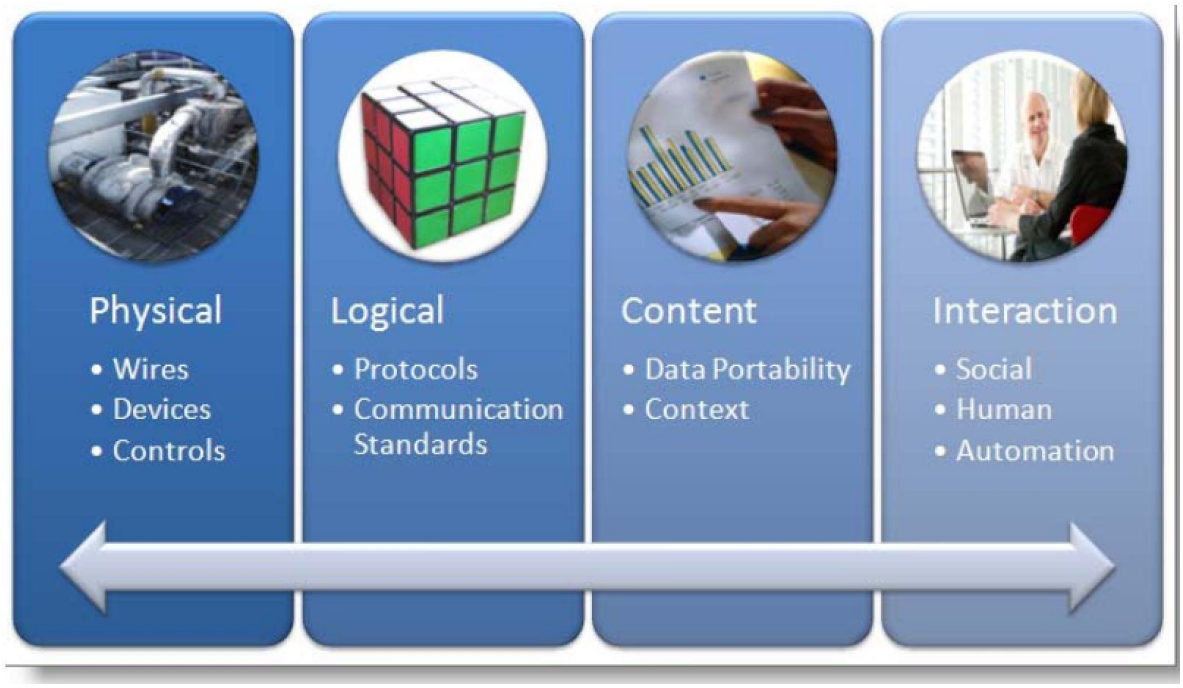

Figure 6: the spectrum of interoperability, from (Sadlier & Laing, 2011) citing (Gasser & Palfrey, 2008).

The same authors pursue their analysis in (Sadlier, Laing, & Shields, 2012) with a focus on data aggregation, as a mean to achieve interoperability. This issue of data aggregation was addressed in a series of papers ((Behounek, et al., 2017), (Behounek, et al., 2017)) where a novel aggregation system was described that removes the need for third parties to implement dedicated drivers for the various communication protocols used on a drilling rig. This is clearly a first important step towards syntactic interoperability. However, the issue of semantic interoperability is rarely addressed: one should note nevertheless an increasing interest in the ability to properly interpret specific drilling signals. Indeed, some special cases have been analyzed in the recent years: downhole signals (Baumgartner, Zhou, & van Oort, 2016), high frequency and mechanical signals (Baumgartner, Ashok, & van Oort, 2019) (Macpherson, Pastusek, Behounek, & Harmer, 2015) and basic traditional signals such as ROP (Goncalves, et al., 2017).

Interoperability in drilling is in its infancy. Some other industries, especially those with a string focus on IT, have come further on the interoperability path. It seems that the initiatives surrounding the emergence of IoT (Internet of Things) technologies have in particular enabled semantic interoperability. The involved technologies are all based on the Semantic Web framework. This initiative from W3C, initiated during the 90's in the team around Tim Berners-Lee, provides programming languages for the representation, manipulation and computation of meanings and knowledge. In the case of IoT, it can be used to properly describe the properties and functions of the different devices under consideration. A recent series of whitepapers (Bauer, 2019) (Bauer, 2019) (Murdock, 2016) provide deep insight into the gains of interoperability that can be achieved through proper

semantical modelling. In particular, (Murdock, 2016) gives a very good overview of those benefits, covering the reduced cost in data integration as a function of the number of data sources (see Figure 7), and the modularity, reusability and flexibility for hierarchical representations of the various semantical models. (Figure 8).

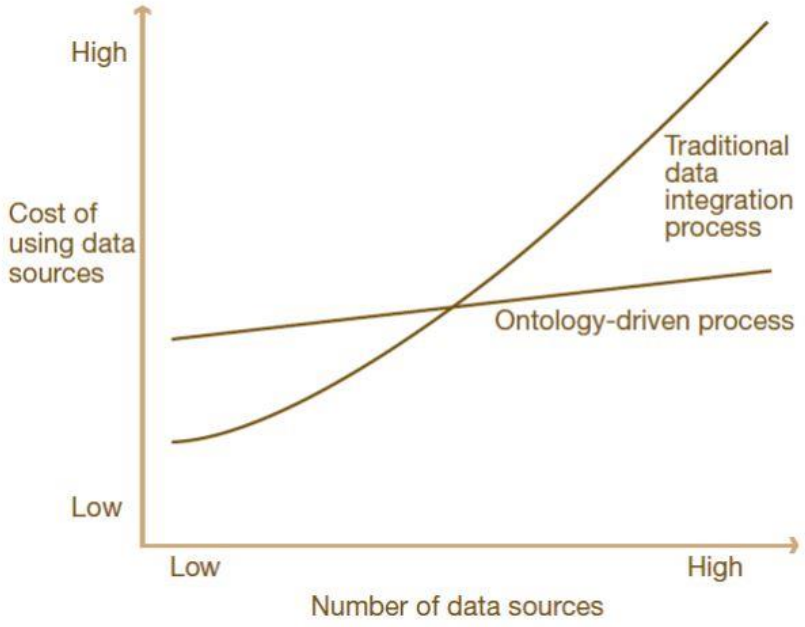

Figure 7: cost associated with integration of data sources, from (Murdock, 2016). Semantical modelling (ontology-driven process in the picture) performs better than classical integration approaches when the number of actors gets high.

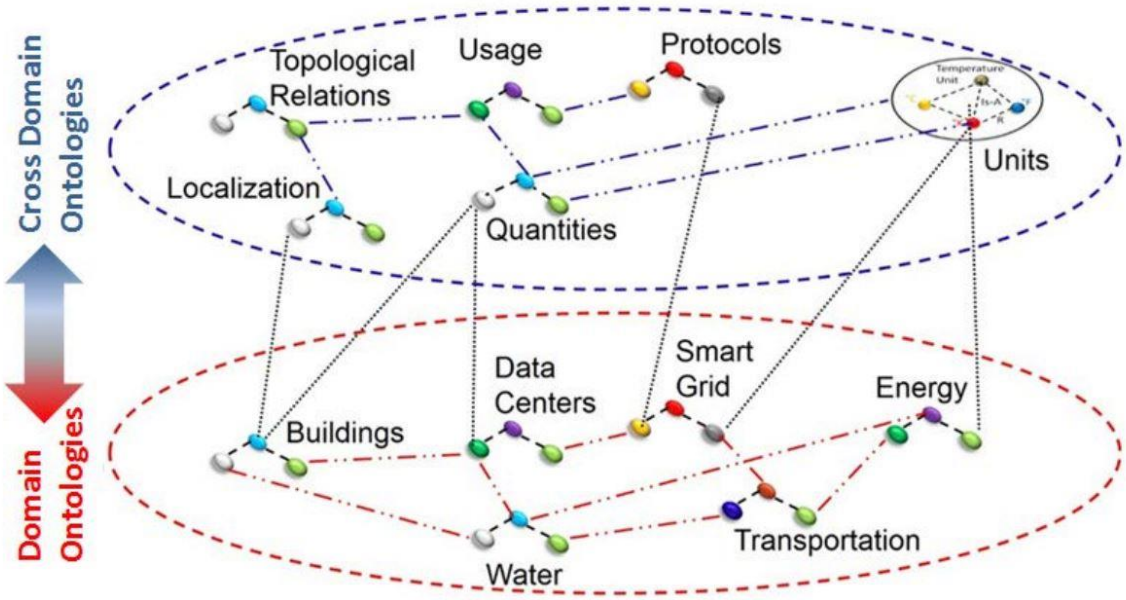

Figure 8: layered semantical modelling (from (Murdock, 2016)). Generic and multi-purpose ontologies can be re-used in very different context such as transportation, energy, water systems, buildings, etc.…

More technical discussion about the optimum design and development of interoperability standard can be found in (Bauer, 2019) (Bauer, 2019). Here several key aspects of the collaborative efforts required to achieve interoperability are discussed in depth, and are of

primary importance for the future of our work. Purely technical aspects are treated in (Bauer, 2019), with detailed examples about Smart homes and Smart Grids, while the development of standards, in their various forms, is discussed in the second reference, (Bauer, 2019).

4. Challenges

If we want to make the data acquisition process as smooth as possible in the future, we need to identify the hurdles that make it cumbersome today. We review below what we believe are the most problematic characteristics of the real-time data infrastructure of a modern drilling rig.

Data set-up variability.

It is unfortunately not the case that a data acquisition configuration can be established once and for all for a given installation. In practice, configuration must be significantly updated for each run and maintained during the course of operations.

One reason for this is the collaborative nature of the drilling operations. Drilling operations are usually performed by a constellation of companies (drilling contractor, downhole service company, mud logging company, drilling fluid provider, cementing services) which all have their specific role and domain of responsibility. Depending on the nature of the ongoing operation (tripping, drilling, cementing) this constellation may change, and this change is reflected in the set and nature of available signals.

When considering the most commonly used signals for drilling automation, namely the top-side rig signals, these are expected to be made available in a rather stable fashion: they are usually provided by companies involved in all operations (drilling contractor, drilling control system provider). There is nevertheless variability in this set of data. For example, top-side sensors are regularly calibrated, and their characteristics change on a regular basis. Some surface sensors are only available in certain conditions: when tripping using the elevator, top-drive sensors are of course inactive. One regularly encounters more subtle situations, such as the unexpected use of cement pumps to support the mud pumps, or the non-reported use of booster pumps in the riser. Such situations can occur at almost any time, and their critical influence on the circulation patterns demands immediate update of the data acquisition system. Finally, it is not uncommon to use different drilling techniques for different sections: one section may be drilled riser-less, while the next may be drilled with a riser and BOP, or one section may be drilled conventionally, while back-pressure MPD can be used for the following section. This results in significant changes in the nature and meaning of the different signals. Such meaning must be communicated together with the signal.

The situation is more dynamic down-hole. The precise number, nature and location of the sensing devices depends entirely on the BHA design and are therefore differs from run to run. When mud-pulse telemetry is used, the bandwidth limitations impose optimized design of the sampling rates for every single record, depending on the operations' local conditions. When distributed sensors are installed along the drill-string, the number of

available signals will be regularly incremented as the pipe is tripped inside the hole and decremented while pulling out. Here as well, data acquisition maintenance procedures are required to adapt to those changing conditions.

Finally, there is no reason to limit the set of real-time data to sensory signals. Indeed, the outputs of advanced monitoring and automation systems are also part of the data ecosystem: drilling optimization recommendations are consumed by the drilling control system, as well as limiting operational values. Simulated sensor values can be used by other agents, for example to detect process deterioration. High flexibility is achieved in the data set-up when including those signals: the involved actors can change during operations; computations can be turned on and off at any time during a run. This must be captured properly by any system that makes active use of the signals.

Multiple sources: refresh rates, delays and synchronization.

As stated earlier, the fact that many companies participate in the well construction process is a source of confusion. From a real-time data perspective, this is reflected in the time behavior of the various signals.

Depending on their origin, signals reach their final destination through different telemetry systems and local networks. This induces noticeable variations in the overall transmission delay: the delay between the actual measurement time and the acquisition by the real-time data management system. The transmission delay itself can vary during operations for downhole signals: the delay depends on the sensor depths via the pressure wave propagation time for mud-pulse telemetry and via the number of encountered transmitters for wired transmission. Timestamping of data usually only considers the acquisition time and depends on the internal clock used by the network that sets the time stamp. It is rather common that different networks use clocks that are not synchronized.

The combination of transmission delays, multiple clocks and timestamping of acquisition time instead of measurement time results in a non-synchronized real-time data set: at any time, the ensemble of available signals corresponds to events that may be from a few milliseconds up to a dozen seconds old. While it seems difficult from a technological point of view to significantly reduce the various delays, the knowledge of transmission characteristics is a requirement to any application that needs to synchronize the incoming signals (Isbell, 2017).

Multiple processing

Drilling signals are seldom raw measurements. Besides the obvious conversion from electrical signal to measurement via transfer functions, several layers of data processing may be applied between the measurement itself and the final acquisition.

While the standard filtering applied to top-side signals does not affect the interpretability of the data, the possibly advanced statistical treatment of downhole signals has a significant impact on its meaning. Indeed, analysis systems need to know if the downhole pressure is the result of a moving average or a simple sample, the window size of a peak-to-peak downhole RPM, or if a downhole WOB has been corrected for gravity effects or not (see (Baumgartner, Zhou, & van Oort, Efficiently Transferring and Sharing Drilling Data from Downhole Sensors, 2016) and (Baumgartner, Ashok, & van Oort, Automated Preprocessing Techniques for High Frequency Downhole Sensor Data, 2019) for a detailed analysis and use of such specifications).

Multi-dimensionality and digital representations

Most measurements take the form of single scalar values and can be unambiguously digitally represented. However, the situation gets more complex when multi-dimensional data needs to be transferred between applications. In such cases, there may exist several possible representations of the same quantities. For example, a three-dimensional velocity can be stored internally as a one-dimensional array with three inputs, or as a three-dimensional array with one entry per dimension. Those very same representations can also be used to store three one dimensional velocities, measured at different places along the drill-string. The recent developments in terms of sensing and transmission technologies now make it possible to access multi-dimensional data measured at different locations: the number of possible internal representations then grows with the number of combinations between the different dimensions involved. Three-dimensional accelerometer readings provide a good illustration of this situation. Continuous rheological measurements, performed at varying temperatures or pressures, is another example.

Without proper knowledge of the representation used by the sensor company, correct interpretation of the digital signal is impossible.

New sensors

Sensor technology is in constant evolution. Over the last decades, advances in fluid properties measurement techniques and downhole mechanical sensors have provided new types of signals of high relevance for automation systems.

There is no reason to believe that the pace of innovation will decline in the future. As a consequence, based on today's data management systems, real-time data consumer applications will have to constantly learn the specifics of each new signal, and continuously update these as technology matures. An obvious risk is that consumer companies do not access these measurements before they are provided as standard drilling data. As a consequence, the new sensors may fail to prove their value to the drilling process because they are not used.

5. Semantic interoperability example

In order to illustrate the importance of semantical information and how it can help reaching full interoperability between different drilling automation systems, we will use an artificial example, as presented in this section.

Over-pull detection

We consider the problem of automatic detection of over-pull, *i.e*. an abnormal increase of hook-load (see Figure 9). Such events can occur rather often during operations. They can be caused by pack-offs (due to the packing of solid particles in the annular, which obstructs the flow-path and creates additional drag forces when lifting the drill-string up), ledges, or hole collapse. In any case, such occurrences require as quick as possible reaction to avoid further damage on the well itself and on the top-side equipment. Overpull may be detected by comparing the measured hook-load to some expected hook-load: this is the classical fault detection approach.

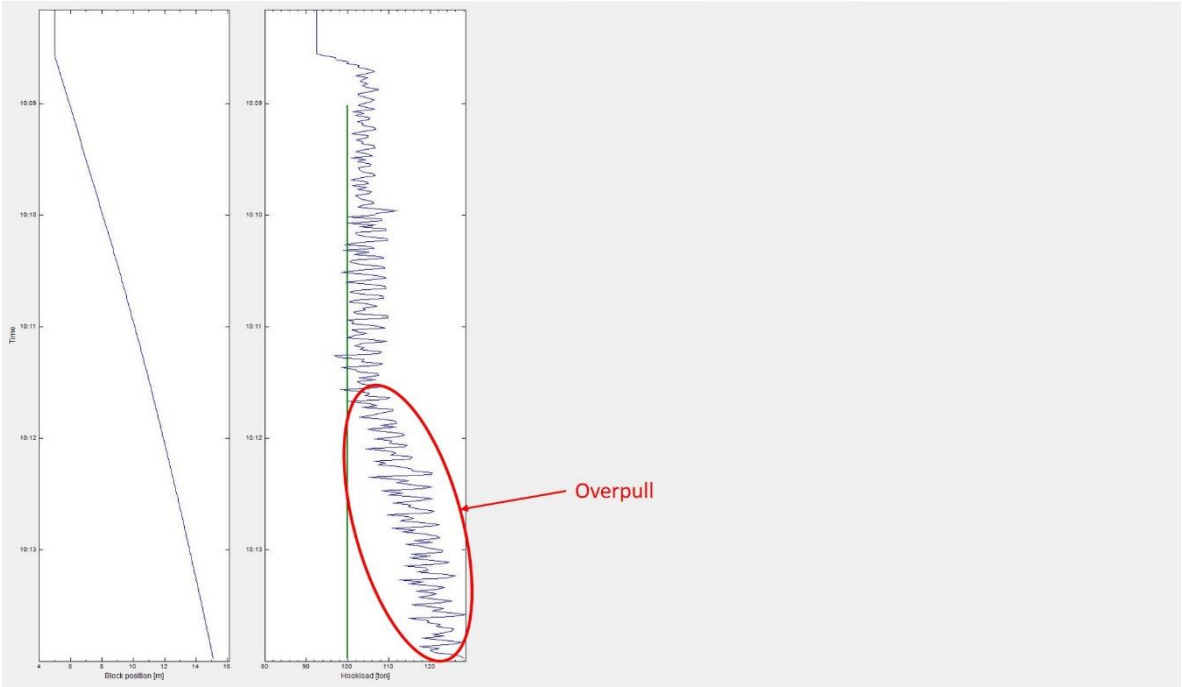

Figure 9: overpull starting at approximately 10:11:30. The left chart shows the block position during a lift of the drill-pipes. The second chart shows the measured hook-load (blue curve), with a clear increasing trend from ca 10:11:40. The green line correspond to some (steady-state) torque and drag calculations indicating the expected hook-load.

We consider in our example three systems that will collaborate to achieve proper detection capabilities:

- RIG: the rig system, that will provide surface sensor data.
- DAS1: the first automation system will provide calculations of the expected hook-load given the current operational conditions (hook velocity, flowrate, surface RPM). The computations are performed using a steady-state torque and drag model and provide the expected load at the top of the drill-string.
- DAS2: the second system has the responsibility of detecting the over-pulls. It will gather all the available information (from measurements and calculations), perform the necessary comparisons and issue notification when significant discrepancies are observed.

We will now explore several data configurations which vary in terms of available semantical information, and see how this affects the efficiency of the detection of discrepancies.

Configuration 1

The first configuration is the simplest one:

- The RIG system exposes[2] 2 signals, tagged as hook-position and hook-load, without additional information. It therefore means that the RIG system notifies the other participants of the existence of those signals, but does not provide any more details about them than their typical drilling function.
- The DAS1 system detects those 2 signals, in particular the hook-position signal which enables the system to calculate an expected hook-load. The result of the calculations is made available to the other participants, namely the DAS2 system.
- The DAS2 system has access to 3 signals: hook-position, measured hook-load and calculated hook-load. Without more information, the only possible strategy to detect overpulls if:
    - The hook is lifted up and the measured hook-load is larger than the calculated one.

Figure 10 shows the results of this strategy applied to the same data sequence as in Figure 9. Because the calculated hook-loads are constantly less than the measured ones, overpulls are detected continuously during the entire sequence, including this period of time when the hook-load seems to behave normally.

[2] By exposing a signal we mean that the system adds a signal to the real-time environment, possibly adds additional information about the new signal, and that the rest of the participants are notified of those additions.

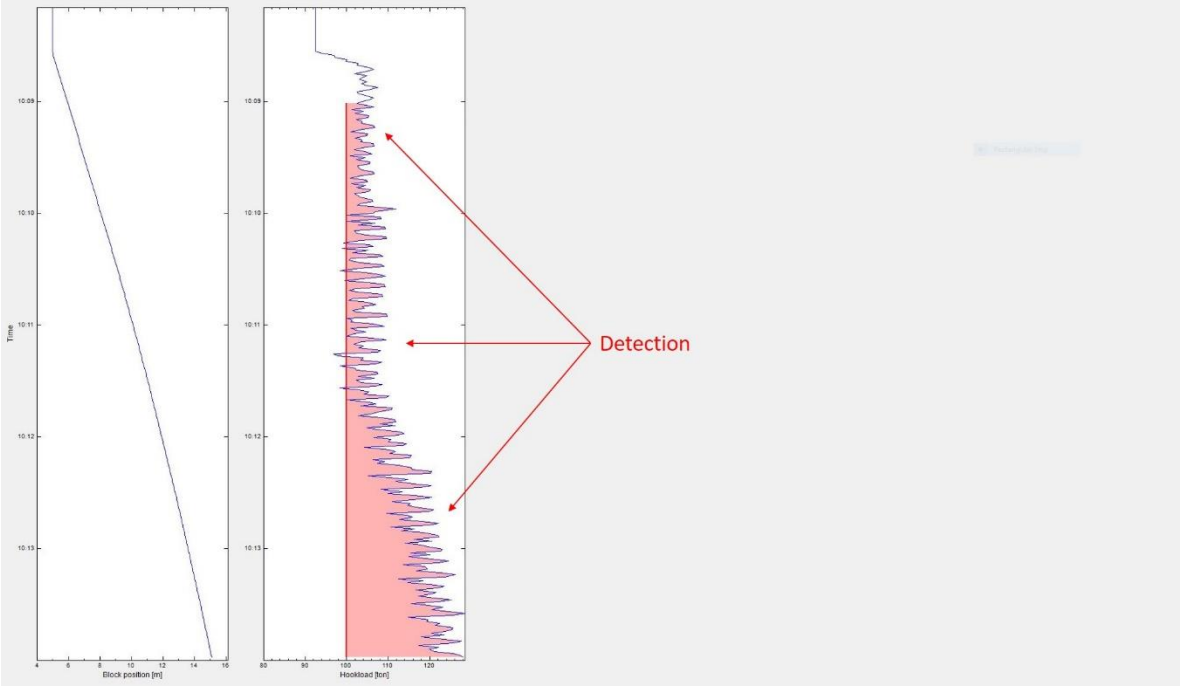

Figure 10: overpull detection in configuration case 1. The detection algorithm only has access to measured and calculated values, without any further information. In this case, overpulls are erroneously detected during the entire sequence.

Configuration 2

The second configuration makes use of additional information, namely the characteristics of the sensor device used to measure the hook-load:

- The RIG system exposes 2 signals, tagged as hook-position and hook-load. It also provides the accuracy of the hook-load measurement.
- The DAS1 system detects the 2 RIG signals. The hook-position signal enables the system to calculate an expected hook-load. The result of the calculations is made available to the other participants.
- The DAS2 system has access to 3 signals: hook-position, measured hook-load and calculated hook-load. It also knows the accuracy of the hook-load measurement. It can now modify its strategy and detect overpulls when:
    - The hook is lifted up and the measured hook-load minus its accuracy is larger than the calculated one.

Accounting for the measurement accuracy makes the detection algorithm more robust. When reading a hook-load measure of for example 100t with a provided accuracy of $\pm$5t, one can be certain that the hook-load is larger than 95t. It is this reasoning that is used: the comparison is now done between the measured hook-load minus the accuracy and the calculated value, as is illustrated in Figure 11.

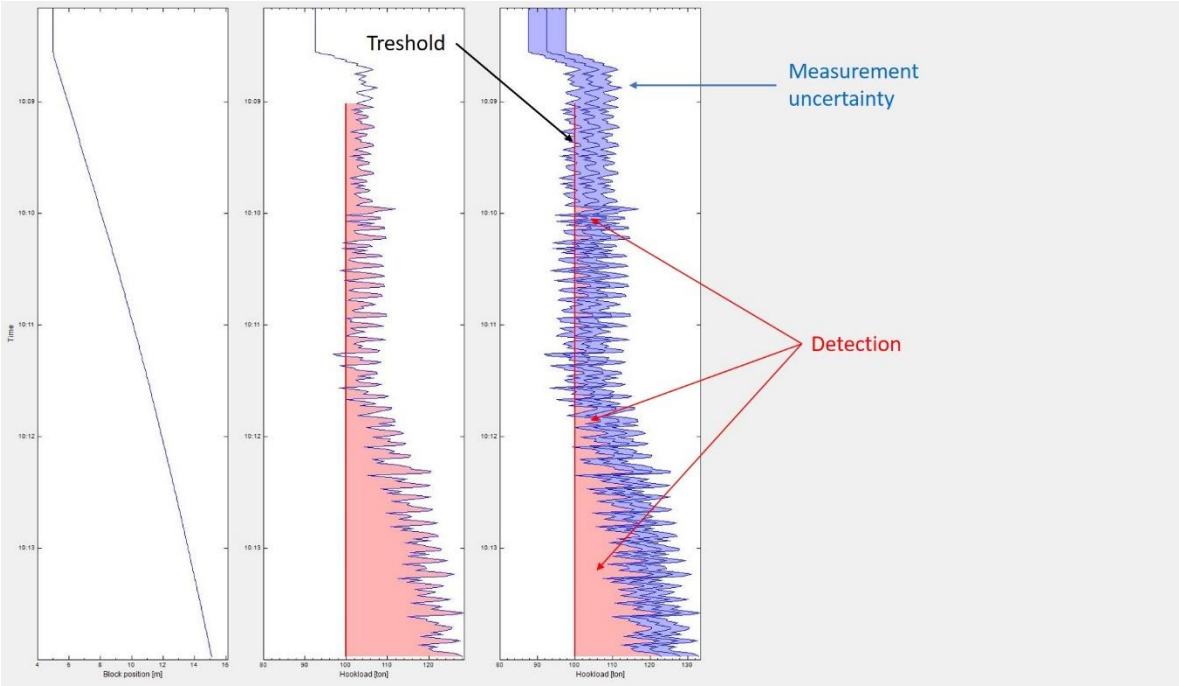

Figure 11: the measurement accuracy is now provided. The detection algorithm uses this information by subtracting the accuracy to the measurement prior to performing the comparison. This results in better overpull detection capabilities.

Configuration 3

The sensor uncertainty is far from being the only information relevant for a measurement. In many cases, the location of the sensor is useful information. In the case of hook-load it is of primary importance. In many instances, the signal is acquired at the deadline. The schematic of Figure 12 emphasizes the different parts of the hoisting system that makes the deadline tension differ from the tension at the hook itself.

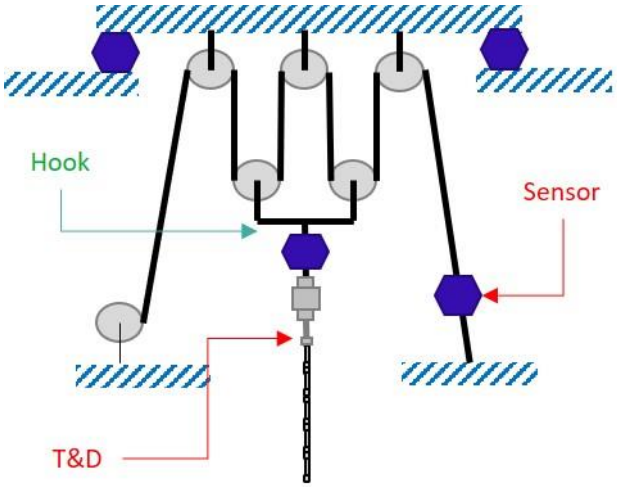

Figure 12: measurement and calculation locations. The sensor is placed at the dead-line, while the calculations made by the torque and drag model only estimate the tension at the top of the drill-string.

In configuration 3, the sensor location (the deadline) is included in the information provided by the RIG system. This added information can be used to automatically trigger changes in the detection approach:

- The RIG system exposes 2 signals, tagged as hook-position and hook-load. It also provides the accuracy of the hook-load measurement and the location of the sensor.
- The DAS1 system detects those 2 signals, in particular the hook-position one that enables it to calculate some expected hook-load. The result of the calculations is exposed to the other participants. In addition to the calculation, the DAS1 system provides the location of the estimated value, namely the top of the drill-string, hereby indicating that it does not account for the hoisting system.
- The DAS2 system has access to 3 signals: hook-position, measured hook-load and calculated hook-load. It also knows the accuracy of the hook-load measurement, and the locations of both measurement and calculations. Based on this location information, the system can estimate modelling uncertainties, accounting for the fact that the hoisting system is not included in the computations (see Figure 12). The uncertainty estimation can be based on empirical observations, or on hoisting system modelling (Cayeux, Skadsem, & Kluge, 2015). The system can now be modified to detect overpulls when:
    - The hook is lifted up and the measured hook-load minus its accuracy is larger than the calculated one plus its uncertainty.

The same reasoning as in the previous configuration applies here, but is extended to the modelling uncertainty: if a calculation predicts a hook-load of 100t $\pm$ 5t, then we know for sure that the calculated hook-load is less than 105t. Therefore, the threshold to use when making comparison is the calculated value plus its uncertainty. This leads to improved detection capabilities, as can be seen in Figure 13.

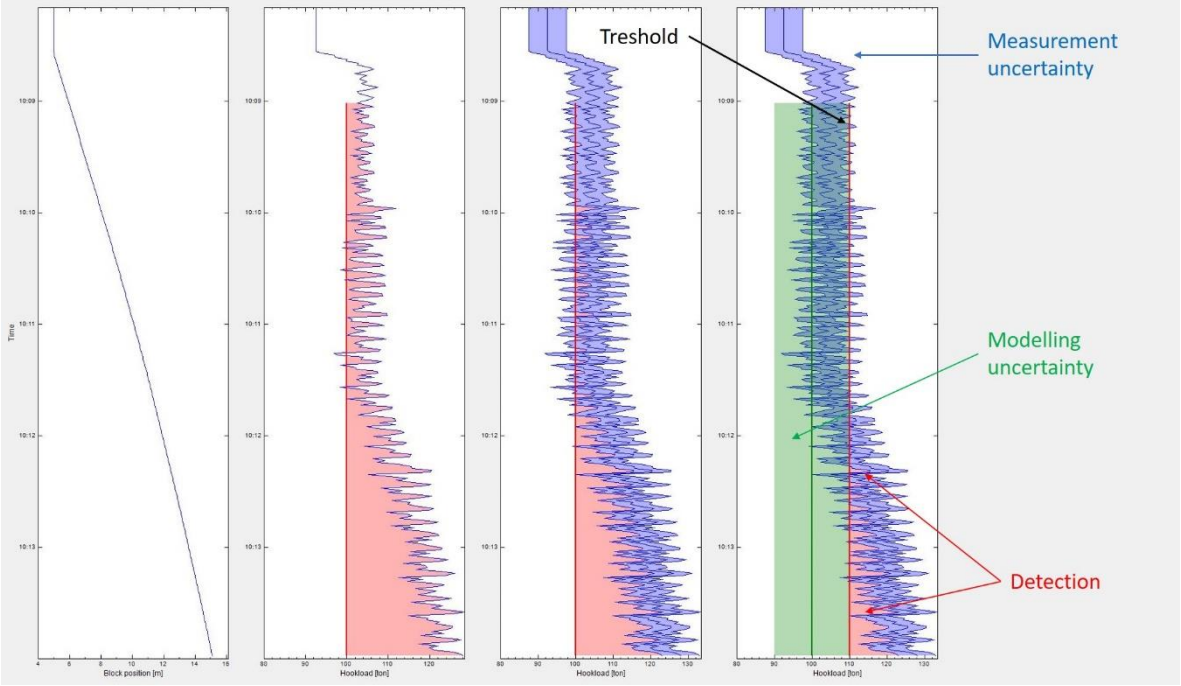

Figure 13: here, the location information (for the sensor and the calculation) is used by the DAS2 system to estimate some modelling uncertainty. In turn, this modelling uncertainty leads to a modified detection algorithm.

Configuration 4

The fourth and last configuration is the one that provides the best result. Here, the RIG system is able to provide two different types of measurements for the hook-load: a load cell has been installed in the top-drive. This signal, as well as its uncertainty, is made available to the rest of the system:

- The RIG system provides 3 signals: a hook-position and two hook-loads. For each hook-load, the measurement accuracy as well as the sensor location is provided.
- The DAS1 system detects the RIG signals, where the hook-position signal enables the system to calculate some expected hook-load. The result of the calculations is made available to the other participants. In addition to the calculation, the DAS1 system provides the location of the estimated value, namely the top of the drill-string, hereby indicating that it does not account for the hoisting system.
- The DAS2 system has access to 4 signals: hook-position, two measured hook-loads and one calculated hook-load. It also knows the accuracy of the hook-load measurement, and the locations of both measurements and calculations. It can apply the same strategy as in configuration 3, but now has to choose between the dead-line hook-load and the top-drive one. Knowing their location is important: since they are supposed to be compared with some calculation, whose location is also known, then the more natural choice is to select the measurement closest to

the estimation In this case, it is the measurement made at the top-drive. The final strategy is then that an overpull is detected when:

- The hook is lifted up and the measured hook-load minus its accuracy is larger than the calculated one plus its uncertainty.

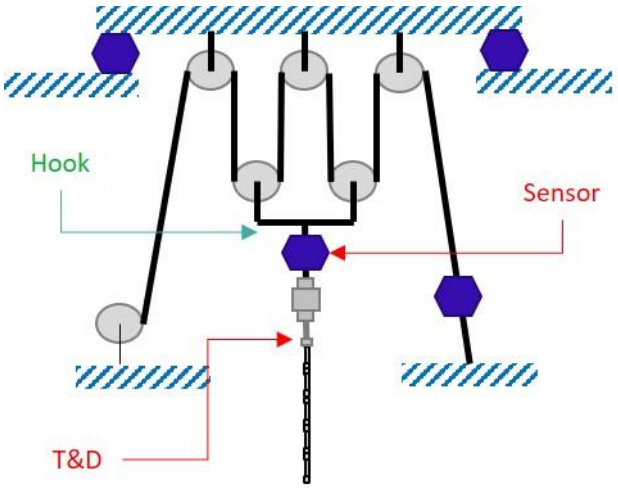

Figure 14: last configuration. An additional signal is available, with measurement acquired at the top of the top-drive. The distance between the sensor location and the estimation location (top of the drill-string) is much smaller than for the previous configuration.

In this latter case, the semantical information is not used to adjust the detection mechanism but to select the optimal source of information: the combination of sensor accuracy and precision was in this situation the relevant criteria for deciding between the dead-line and top-drive hook-loads.

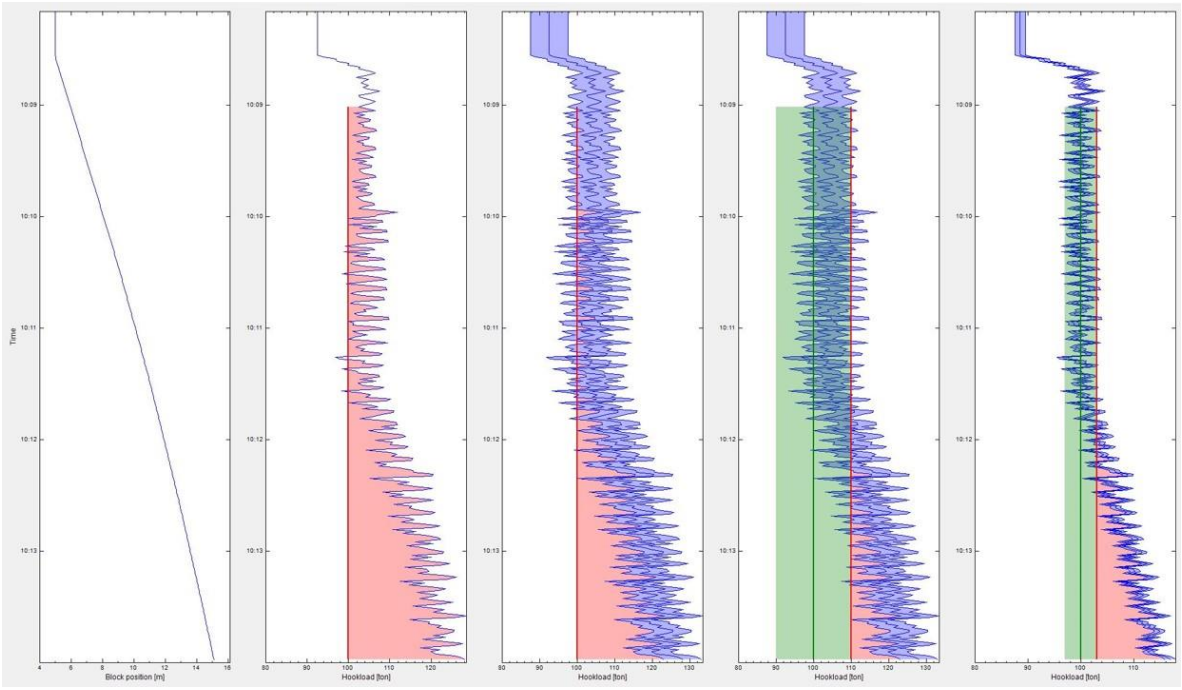

Figure 15: the two tracks on the right show the final detection strategy (i.e. including measurement and modelling uncertainty based on locations) for the two available hook-load sensors. The fact that the top-drive hook-load is better suited for this application can be deduced from the semantical information.

Conclusion

In this artificial example we saw how contextual information can significantly improve even simple automation systems. Several aspects of interoperability have been addressed:

- Here, the system as a whole achieved much better results when each component provided as many details as possible regarding the data it delivered.
- When the semantical information can be interpreted by a computer system, the treatment of the contextual data can be automated so that automatic system configuration becomes feasible.
- The automatic configuration functionality requires some preliminary work. In our case, the DAS2 system performs several adjustments based on the available information: estimation of modelling uncertainty, modification of threshold values for comparisons, etc. These features need to be considered in the design phase of the drilling automation system, and therefore require dedicated work. However, this preparatory task is done once and for all: the same adaptation mechanisms are valid for any rig that is capable of delivering semantical information. This in contrasted o the current conventional approach, where very little focus is put on adaptability (because the contextual information is anyway unavailable) and for which every function installation requires significant ad-hoc adjustments with applicability is limited to the specific installation.

- Semantical information facilitates correct interpretation of measurements but also of computations: in an industry that relies more and more on automation solutions, simulations are playing an increasingly central role in the decision process. Many automation solutions rely on processing of both measured and calculated data, with the latter originating from data driven techniques, numerical models or hybrid solutions. In any case, such calculations are a source of information as valuable as real-time measurements and deserve therefore the same attention as them with respect to interoperability.

6. Data and drilling data modelling

One of our objectives is to enable a computer friendly description of the drilling data available at the rig site. In other words, we wish to develop a framework for drilling data knowledge representation. There exist several approaches for this purpose, and the various techniques cover a broad range of expressivity and flexibility, sometimes called the *semantic spectrum*.

At the lower end of the semantic spectrum one finds glossaries, *i.e.* collections of names with definitions, that define the entire set of data one may encounter. The WITS0 data transfer standard (see (WITS, u.d.)) corresponds to such a glossary: the list of signals is predefined and associated with a definition that provides little information on details of the data. This simple approach suffers from several limitations: when considering complex data, the number of entries grows exponentially with the number of parameters, while the glossary is fixed and any modification has to be communicated to all the users; further, the approach is not suited for expressing the numerical quantities that are essential for the description of the data.

Taxonomies (or containment hierarchies) are a slightly better alternative: they correspond to tree-like classifications where each composite data type or object can contain specific fields (Booleans, integers, floating points…). Although they are an improvement to glossaries, they are still too restrictive: their inheritance tree structure is not suited to describe data along several semantical axis (processing, time management). Borges' famous taxonomy (Borges, 1942) illustrates the difficulties of complex data classification.

In order to circumvent this problem, modern object-oriented programming languages allow cross references between object types as well as multiple-inheritance (C++) or the use of interfaces (C#, Java), effectively turning the inheritance tree into a graph. The WitsML data model (Energistics, u.d.) is an example of this approach applied to drilling related information. However, as different programming languages have various approaches to solving the generic problem of multiple inheritance, the portability across multiple programming languages of a semantic definition of drilling real-time signals is restricted. In addition, one should note that the WitsML data type structure is static, meaning that any extension of the data model requires a revision across the whole industry.

At the upper end of the semantic spectrum are true semantical models. Semantic nets were formally introduced in the 50's for natural language processing (see (Simmons, 1963)). They are graphical representations of domain knowledge: relational facts are expressed by triplets (sentences) *<s,v,o>* were the subject *s* and the object *o* are related by the verb *v*. A set of such sentences can then be represented as a graph whose nodes are the different objects and subjects, and the edges (connectors) are the verbs (relations). A simple graph is shown in Figure 16). In this context, semantical modelling consists in defining the different

data or relation types that nodes and edges can have. Modern knowledge representations extend this framework with logical definitions, and define advanced specifications and rules for constructing and extending/manipulating semantical models. Construction and development of such models is in this context equivalent to creation and manipulation of ontologies[3]. The most common language to create and manipulate ontologies is OWL (and its variants such as OWL2, OWL-DL), built on the RDF framework, which forms the basis of the Semantic Web technology stack and architecture. Note that when creating ontologies, the same formalism (specifications and rules) is used to specify the data type structures (type inheritance, attributes) and the instantiations specific to a particular asset. This entails that a computer system can interpret dynamically new type definitions, and the need for industry-wide revisions when extending a model is not required anymore.

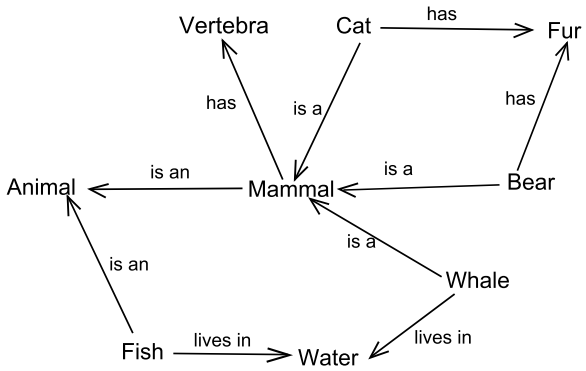

Figure 16: an example of a semantic network. Courtesy: Wikipedia (https://en.wikipedia.org/wiki/Semantic_network)

We describe below some of the existing data models that have some relevance or resemblance to our project.

Wits

Wits (Wellsite Information Transfer Specifications) is probably the ancestor to all the data exchange systems on the rig site. It is based on a simple ASCII representation of the data, coupled with a simple transfer protocol (initially based on Telnet, but updated to support more modern infrastructures with TCP transfers).

The underlying data model is simple, organized in records, each containing a sequence of values. To each index in the sequence of values is associated a predefined meaning, such as Hook-load or SPP. By using a fixed predefined structure, some basic semantic is then carried: each signal is perfectly identified, although it is impossible to provide additional

[3] Wikipedia: In computer science and information science, an ontology encompasses a representation, formal naming and definition of the categories, properties and relations between the concepts, data and entities that substantiate one, many or all domains of discourse.

information to the initially specified semantic. It is for example not possible to specify the location of the hook-load sensor.

The Wits exchange system is still widely used at the rig site, because of its simplicity and relative efficiency: there is little overhead associated with a Wits channel, and therefore relatively high frequencies can be achieved. However, for communication outside the rig site itself, other technologies are used.

WitsML

The WitsML (Wellsite Information Transfer Standard Markup Language) standard is often considered as a successor to the Wits protocol. It provides XML based modelling of drilling data, including extensive covering of configuration data: wells, wellbores, rig, drill-string, architecture. The transfer of real-time data is performed mostly through the use of logs, that are generic structures to represent time data. The genericity of the model allows the representation of a large variety of data, but there is no possibility to properly describe the nature of the signals: apart from units, no structured metadata is available. The identification of the different signals is also problematic: each trace in a log is identified by a mnemonic, and it requires human interpretation to correctly identify the different ones.

Finally, the verbose structure (because of the xml representation) and the log-based storage of data makes the performance of WitsML unsuitable for control applications. Note however that recent developments (in particular the Energistics Transfer Protocol - ETP) may improve the performances of the data transmission.

OSDU

Hosted by the open group (https://www.opengroup.org/osdu/forum-homepage), the OSDU (Open Sub-surface Data Universe) initiative focuses on the modelling of sub-surface data and well data, and is therefore more relevant for exploration applications.

OPAF

This initiative, the Open Process Automation Forum, is also hosted by the open group (https://www.opengroup.org/forum/open-process-automation-forum). Its main concern is to provide standards for automation purposes.

ISO15926

This standard, hosted by the ISO organization, https://www.iso.org/standard/29557.html, aims at specifying a conceptual data model for the computer representation of technical information about process plants, and targets therefore the downstream part of the oil and gas industry.

Quantities, Units, Dimensions and Data Types Ontologies (QUDT)

The QUDT organization is a member of the W3C consortium whose mission is, according to their web-site (http://www.qudt.org/) to "improve interoperability of data and the specification of information structures through industry standards for Units of Measure, Quantity Kinds, Dimensions and Data Types". They therefore distribute ontologies (available in RDF, the Semantic Web base format) covering data-types, units, quantities and vector dimensions.

Spatial Data on the Web

Several ontologies were developed by the Spatial Data on the Web Working Group (https://www.ogc.org/projects/groups/sdwwg), a subgroup of the OGC Geosemantics DWG (https://www.ogc.org/projects/groups/semantics), itself part of the Open Geospatial Consortium (https://www.ogc.org/). It works in parallel with a group of the same name (SDWWG), affiliated to W3C (https://www.w3.org/2015/spatial/wiki/Main_Page).

Their work has resulted in:

- The Semantic Sensor Network Ontology (SSN), available at https://www.w3.org/ns/ssn/. According to their web site (https://www.w3.org/TR/vocab-ssn/), it is an ontology for describing sensors and their observations, the involved procedures, the studied features of interest, the samples used to do so, and the observed properties, as well as actuators.
- The Sensor, Observation, Sample and Actuator (SOSA) ontology: a lightweight and self-contained ontology containing the elementary classes and properties used by SSN (see (Janowicz, Haller, Cox, Le Phuoc, & Lefrancois, 2019)).
- OWL Time Ontology (https://www.w3.org/TR/owl-time/). According to their web-site, this ontology provides a vocabulary for expressing facts about durations and about temporal position including date-time information (see Figure 17).

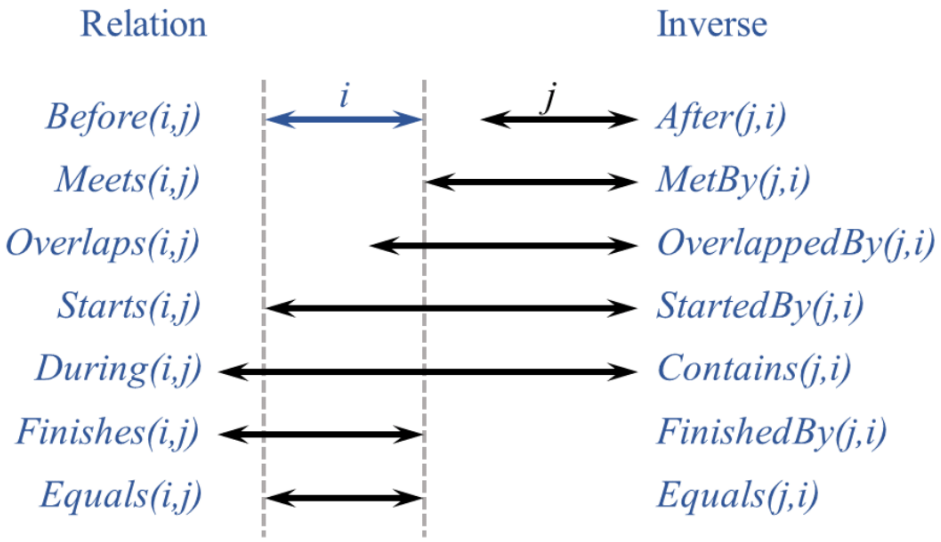

Figure 17: possible relations between time periods, from the OWL Time Ontology (https://www.w3.org/TR/owl-time/)

Observations and Measurements (OandM)

This is a standard developed by the OGC https://www.ogc.org/standards/om. More specifically, this standard defines XML schemas for observations, and for features involved in sampling when making observations.

Smart Energy Aware Systems (SEAS)

This is an ontology built upon the SOSA and SSN ones, devoted to Smart Energy Systems: https://ci.mines-stetienne.fr/seas/

SensorML

Developed by the OGC (https://www.ogc.org/standards/sensorml), it is a standard for defining processes and processing components associated with the measurement and post-measurement transformation of observations.

OBDA

Ontology Based Data Access is not a data model but rather a methodology for querying databases via ontologies. It can therefore be seen as a technique for seamless integration of several different data bases, and has been used as such in the oil and gas industry ((Kharlamov, et al., 2017), but also proved useful when integrated with real-time data in other contexts (for example at Siemens (Kharlamov, et al., 2017)).

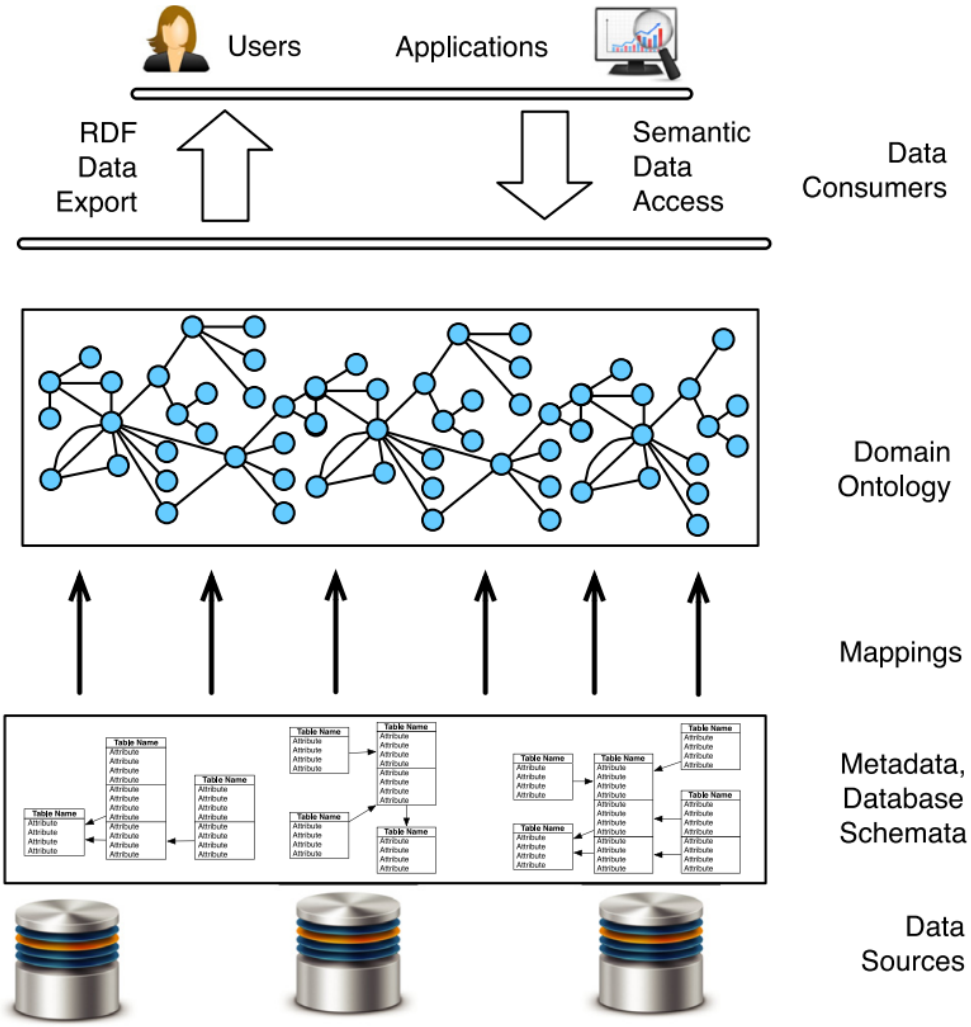

Figure 18: general diagram of OBDA, from (Kharlamov, et al., 2017). Domain ontology pictured as graph.

7. Definitions

As an attempt to solve the interoperability problem, the DDHub provides tools to the drilling community to communicate the real meaning of the available real-time data, in a machine friendly format, so that the time-consuming step of automation system configuration can be automated.

These tools fall into several categories, as they enter the rig data flow ecosystem at different places. We review in the following sections the different components of the DDHub

Semantical model

The main delivery of the DDHub project is a semantical model. Both theoretical and practical aspects of the model will be presented in section 9. At this stage of the report, it is sufficient to note that the model will consist of:

- A list of words, to be shared across the industry by all the involved actors. This list contains all the terms that can be used to properly describe the various signals available, the relations between the various signals, as well as the elements required for this description (such as sensors, signal providers, quantities, computation units…). Each word in the list comes with an associated definition: as such, this list can therefore be seen as a dictionary, but we prefer to refer to it as a vocabulary, as it is the term classically used by the semantical modelling community. This delivery is therefore called the *DDHub vocabulary*.
- A mechanism to construct sentences from the DDHub vocabulary. In natural languages, not all combinations of words are accepted: one relies on a grammar to build valid sentences, intelligible by other individuals. Similarly, not all words belonging to the DDHub can be linked to each other by any other world. The rules guiding the sentence construction form what is called the *DDHub grammar*. A set of sentences built using the DDHub vocabulary and grammar is called a *DDHub graph*
- A set of rules guarantying a minimum validity of the sentences built using the grammar and the vocabulary. The grammar will for example prevent users from writing sentence comprising only verbs and will only accept grammatically correct sentence assertions. However, without further rules, faulty assertions may be constructed. Asserting that a SPP signal is measured by a Coriolis flowmeter should for example be prohibited. The set of rules to be followed so that no obviously wrong sentences are built is called the *DDHub rules*.
- A query mechanism, that allows a user application to search for relevant data within a DDHub graph. This query mechanism is built upon descriptive logic and, although it is by no means normative (other query mechanisms can be used on a DDHub graph), we use it to exemplify the potential of semantical modelling for drilling

automation. We call the set of queries built using this formalism the *DDHub concepts*.

The first three components of the semantical modelling framework are normative: any actor wishing to use the DDHub for their application or real-time data infrastructure has to comply with the syntax and rules defined above. The latter point, the DDHub concepts, is not prescriptive: the descriptive logic formalism underlying the query system is only a possible approach for semantical reasoning, and our proposal indeed only one among the many possible implementations of descriptive logic-based reasoning. There exist several alternatives that are just as valid, or computationally as efficient, that a practitioner can adopt within the DDHub framework.

Implementations

The definitions above provide instructions for properly describing the drilling real-time data. As such, they can be seen as formal requirements. So far, we have not specified how the "real" construction, editing, and storage of a DDHub graph should be performed. Any system providing such functionalities is called a *DDHub implementation*. Instead of imposing a specific implementation, we will present the required functions such a system should have, and propose a series of implementations for several representation technologies, coined as *reference implementations*.[4] Since we define the possible implementations via a functional view, they can be explicitly described as interfaces between the storage system (*DDHub server*) and the application accessing the semantical model (*DDHub client*), either for reading or writing semantical information. For the sake of clarity, we also distinguish between the semantical information flow and the real-time data flow: we therefore define a *Real-time server* as being a storage and access system of real-time drilling operations, and a *Real-time client* as an application accessing (either for reading or writing) a Real-time server (see Figure 19).

[4] Note that the a priori obvious dichotomy between specification and implementation is in fact blurry: classical semantical modelling systems (such as RDF/RDFS or OWL) allow the writing of both vocabulary, grammar, rules and instances in a unified framework.

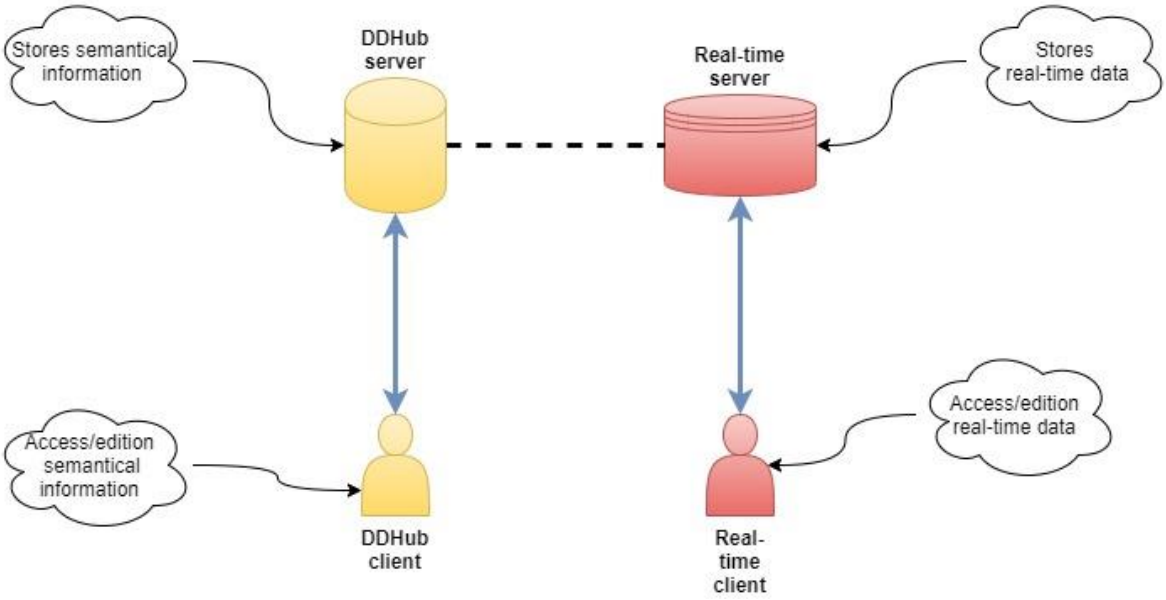

Figure 19: The main components of the signal infrastructure: servers (DDHub and real-time) and clients (DDHub and real-time).

Roles and functions

For the sake of precision, we define 6 roles a client can have (the roles are not exclusive of each other). The roles reflect the type of interaction the client has with the DDHub or real-time servers:

- *DDHub-S producer*: for DDHub semantical producer. A DDHub client that adds or removes nodes in a DDHub graph stored on a DDHub server.
- *DDHub-S provider*: a DDHub client that modifies existing nodes in a DDHub graph
- *DDHub-S consumer*: a DDHub client that loads, reads, or explores a DDHub graph stored on a DDHub server
- *DDHub-RT producer*: a Real-time client that instantiates (exposes) new signals (entries) in a Real-time server
- *DDHub-RT provider*: a Real-time client that writes the values of existing entries in a Real-time server
- *DDHub-RT consumer*: a Real-time client that reads values stored in a Real-time server.

DDHub interfaces

Parts of the project's activities consisted in implementing several interfaces reflecting the different activities related to the different roles defined above. Some of the interfaces relied on native technologies (typically OPC-UA), while others were developed using customized coding.

As illustrated in Figure 20, there are two types of interfaces that are considered: one that allows communication with the semantical part of the implementation, and one dedicated to the real-time data aspect:

- *DDHub source accessor* interface: enables communication between a Real-time client and a Real-time server through some DDHub based syntax. Note that direct communication using native protocol can replace the DDHub source accessor interface. But the DDHub source accessor interface provides standardized data access functionalities, which can prove beneficial for ease of development and installation. The interface is used when playing one of the following roles:
    - *DDHub-RT producer:* the interface provides methods to add (expose) or remove real-time entries in the Real-time server.
    - *DDHub-RT provider:* the interface provides methods to modify the values of the different real-time entries, including time stamping information.
    - *DDHub-RT consumer*: the interface provides methods to read the values of the real-time entries in the real-time server.
- *DDHub adaptor* interface: enables communication with the DDHub server. The interface is used when playing one of the following roles:
    - *DDHub-S producer:* the interface provides methods for adding and removing new items in the DDHub-server.
    - *DDHub-S provider:* the interface provides method for editing and modifying existing elements in the DDHub-server.
    - *DDHub-S consumer:* the interface provides methods to access the semantical model, and to subscribe to changes performed by other actors.

More details will be provided when describing the interfaces implementations, in chapter 10.

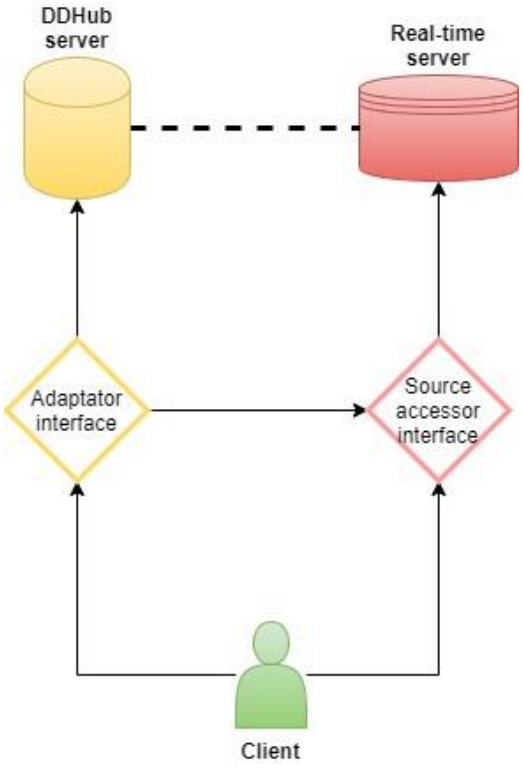

Figure 20: Interfaces

8. Typical scenario

A drilling automation system typically plays the six roles defined above at different stages of its integration within the rig data ecosystem. An integration protocol would classically consist of the following steps (see also Figure 21):

1) Initialization:
   a) Loading of the available signals' semantical description (DDHub-S consumer through the DDHub Adaptor interface)
   b) Identification of the signals required by the system to function properly
   c) Identification of the signals the system will write (if present)
   d) If absent, adding real-time entries in the Real-time server (DDHub-RT producer through the DDHub Accessor interface), and adding the corresponding semantical entries in the DDHub server (DDHub-S producer through the DDHub Adaptor interface): Exposing the semantical entry
   e) Editing/ modifications of the semantical description of the signals the system will write (DDHub-S Provider through the DDHub Adaptor interface).
2) Running
   a) Reading of the input signals (DDHub-RT consumer through the DDHub Accessor interface)
   b) Internal processing
   c) Writing of output signals (DDHub-RT provider through the DDHub Accessor interface)
   d) Monitoring of the DDHub server to detect any change in the real-time signal set-up (DDHub-S consumer through the DDHub Adaptor interface).
      i) Partial repetition of steps 1)a) to 1)e) if changes are detected.
3) Disconnection
   a) Removal of semantical description of the signals the system writes in the DDHub server (DDHub-S Producer through the DDHub Adaptor interface)
   b) Removal of real-time entries in the real-time server (DDHub-RT Producer through the DDHub Accessor interface).

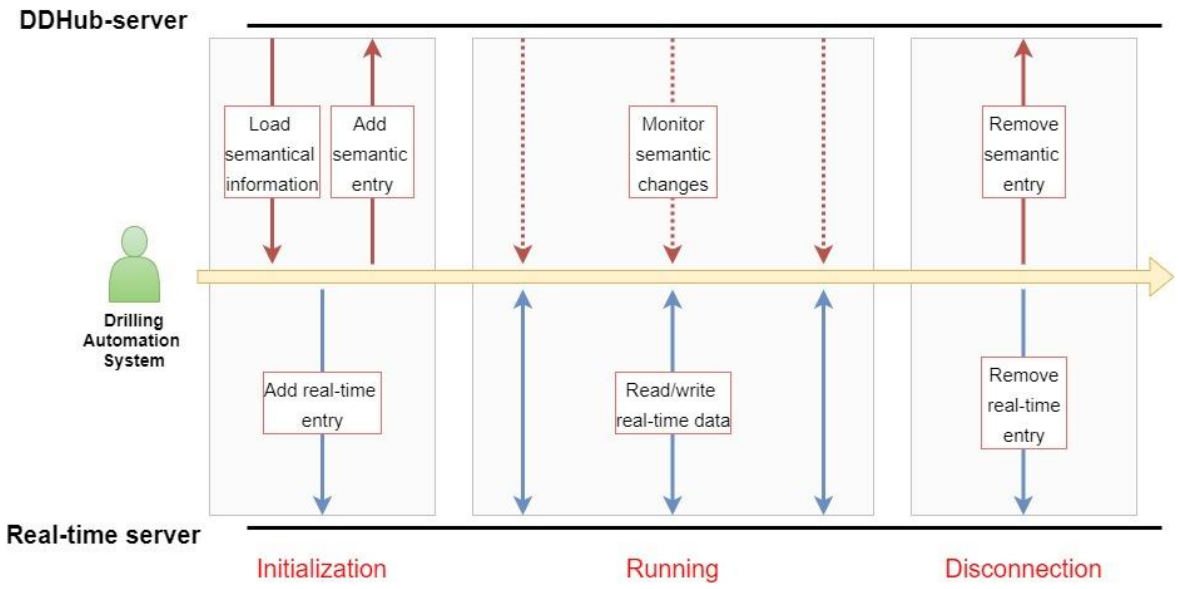

Figure 21: illustration of the typical connection scenario.

Of course, one can imagine alternative scenarios, but the one sketched above contains the main steps that any system would have to go through to get connected to the rest of the real-time data. Note that all of the steps are automated and do not require any human intervention, which is the requirement for full interoperability. In particular, this scenario is repeatable for different installations at no cost: once the work required for its implementation is done, there is no need for additional ad-hoc modifications when deploying the system on a new installation. Note finally that for an automation system to fully utilize the full potential of the DDHub framework, a significant amount of work has to be done to implement the steps 1)b) to 1)e). This work depends on the system's own logic and requirements and necessitates to consider, from the design phase of the system, what are the real-time requirements for the application to run and what information the system provides that can be used by other actors in the drilling process (1)c) to 1)e)).

9. Semantical modelling

The DDHub semantical model consists of a vocabulary defining the possible types and relations that can be used to described the drilling data. This vocabulary is described in this section. The formalism used to construct the vocabulary is first detailed: it defines what are the main characteristics of a type definition. We then introduce the DDHub rules: they can be seen as grammatic rules that have to be respected in order to build conformant graphs.

We give finally an overview of the main topics treated by the DDHub vocabulary.

Formalism

Classical semantical systems (such as RDF, RDFS and their extension OWL2 or OPC-UA) are built a solid theoretical basis. This robustness is achieved by defining a universal framework where both individuals (the "real" objects that are described) and classes (the categories to which individuals may belong) are defined using a unique framework. As a consequence, the classical (in the object-oriented sense) difference between an instance (i.e. individual) and a class is somehow blurry. This can be seen for example in the serialization mechanism[5] of OPC-UA: both types definitions and object instances are serialized into a "node set" file, while in object-oriented programming one would serialize the type definitions into a xsd schema and the instances into an xml file. Of course, the need for distinction between classes and instances is fully acknowledged by the systems, which provide specific means to express the duality, for example by using the rdf:type predicate in RDF. Nevertheless, one still has to construct sentences of the form "Class definition MyClass is an instance of the Class Definition class".

In DDHub the situation is rather different: we assume a clear dichotomy between the formal semantical requirements and their implementation. This gives us the possibility to explicitly distinguish between data type definitions (parts of the semantical requirements) and the instances themselves (as treated in a particular implementation). We use this possibility in our formalism to avoid the many semantical layers necessary to specify whether the considered item is a class definition, or an instance of a class defined in the very same model. All the considered items have a definition function: they specify the attributes that the different types can or must have, and the possible relations between instances. Note however that the chosen formalism can be translated into OWL2 ontologies: this means that we do not lose any of the theoretical robustness of classical semantical systems.

The DDHub semantical framework is a standard one: all the knowledge representation about the drilling signals is constructed using sentences of the form <subject, predicate, object>. Such a sentence can be seen as an edge (the predicate) between two nodes (the

subject and the object). Therefore, a set of sentences will correspond to a (possibly separated) graph, hence the DDHub graph denomination.

More formally we consider two kinds of individuals: types and instances. A *type* is a definition of the common attributes of a set of individuals. It corresponds to the notion of class in object-oriented programming and is commonly called type in the semantical engineering community. It can therefore be seen as a purely abstract notion. An *instance* is a construction of a type entity. We say typically that the individual A is an instance of the type T: individual Robert is an instance of the class of Humans.

DDHub types can be related to each other via inheritance relation. In this context, inheritance means specialization. When a type A inherits from type B, it inherits all its attributes (see discussion below). Inheritance is a transitive relation, but circular inheritance is prohibited. Hence, the structure underlying all the types is a graph. We consider two main families of types: *Literal* types and *Identifiable* types (these two types are the two root types of the DDHub type tree). Literal types correspond to the standard data types (integers, Booleans, floating points, guids), aligned with the classical XSD formats. Any number, string, or date stored on a DDHub server is therefore an instance of a type that inherits from the base Literal type. Identifiable types correspond to complex structures: one can define additional attributes that all instances automatically acquire. Such attributes are called *DDHubField*: a DDHubField is the combination of a Name (of type string) and of a Literal type (typically integer, string…). The attributes defined for the Identifiable type are:

- *Name* (type string): the name of the instance
- *Id* (type Guid): the unique identifier of the instance.
- *Description* (type string): a string allowing natural language description of the class
- *Folder* (type string): meta information that specifies the semantical topic that the given item covers.

Thus, these attributes are defined for all DDHub types. We further split the possible DDHub types into *NodeType* and *RelationType.* The following attributes are defined for node types:

- *AdditionalFields* (collection of DDHubFields): the fields defined by the type. Therefore, any instance of this type will have all the inherited fields plus the ones defined by the AdditionalFields attribute.
- *DefaultValues* (collection of *DefaultValuePair*): a DefaultValuePair is the combination of a DDHubField and of a literal, i.e. an instance of the literal type corresponding to the DDHubField's value. This automatically specifies the value of the field for all instances of the considered type.
- *IsAbstract* (type Boolean): specifies whether the type can be instantiated. If not (IsAbstract = true), then instantiation can only occur for a type that inherits the current type.
- *ParentType* (type DDHub type): the type the current type inherits from.

Relation types define the following attributes:

- *Domain* (type DDHub type)
- *Range* (type DDHub type)

The set of all the DDHub types (including thus node and relation types) for the compound drilling system / environment forms the DDHub vocabulary. This set can be represented in the form of an XML document, but can also be translated into more classical semantical representation systems, such as OWL, and then exported to persistent data (i.e. data that can be stored and exchanged between parties) via the relevant serialization mechanism (N-Triplets, Turtle..).

A *DDHub triplet* **T** is a combination of the form <**s**,**p**,**o**> where:

- **p** is an instance of a type that inherits from RelationType
- **s** and **o** are instances of types that inherit from NodeType. In particular **s** has to be of a type that inherits from the Domain type of the relation **p**, and **o** has to inherit from the Range type of the relation **p**.

This simple rule defines the DDHub grammar. A set of DDHub triplet forms a DDHub graph.

General structure of the vocabulary

Any complex type defined in the DDHub vocabulary has an attribute called folder. This folder is used to categorize the types of information of the DDHub semantics. We have so far identified 10 such categories, all sub-categories of a common “DrillingDataSemantics” generic group (see Figure 22). Each category contains a set of NodeTypes and RelationTypes, constructed such that when two types are related to each other by inheritance, they belong to the same group.

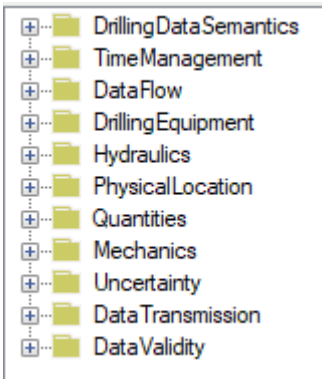

Figure 22: The ten folders, plus the parent folder DrillingDataSemantics.

We will use this folder structure to review the main types defined by the DDHub vocabulary. In this report, we only review the main principles behind the modelling of each topic, using simple examples to describe those principles. A complete description of the vocabulary can be found in the appendix.

Drilling Data Semantics

The Drilling Data Semantics folder is the main folder, containing the central structures. The signals present in a drilling data management system do not all have the same status dynamic. Some are by essence dynamic, such as sensor readings, while some other are of a more static nature, such as machinery limits for instance.

In order to express these differences between signals, we choose first to distinguish between the nodes that contain the semantical data description (nodes called DrillingData), and the nodes that store the signal values (nodes called DrillingSignals). The latter type has a subtype DynamicDrillingSignals, that indicates that the value is expected to change on a regular basis. The DrillingData and DrillingSignal instances can be connected together by relations of type HasValue, which is further subtyped into HasStaticValue and HasDynamicValue. Through such subtyping one can distinguish between configuration data and real-time data.

This approach also allows data that is not associated to any signal, enabling advanced contextualization. Indeed, by doing so one can for example specify that two downhole signals, transmitted via different telemetry systems, and subject to different signal processing, stem for the same original sensor reading. Although the original readings are not present in the real-time data management system, the abstract data is nevertheless represented as it is important to convey such information.

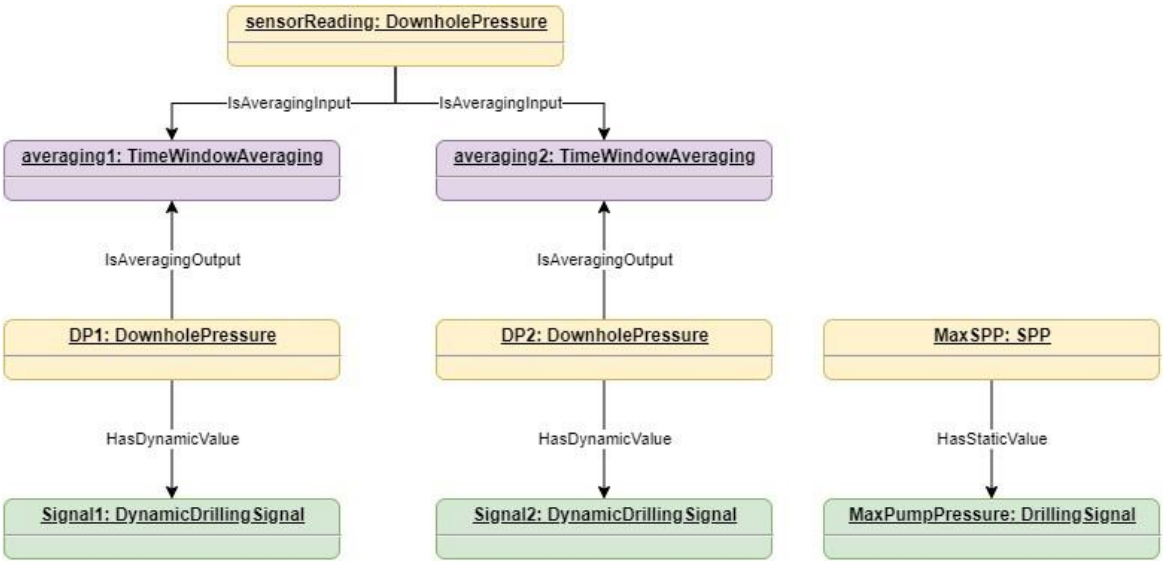

Figure 23: example of data and associated signals. On the right, the maximum pump pressure is a static configuration parameter. The signal is of type DrillingSignal. The node that stores the semantical description is of type DrillingData (in green), and the nodes are connected by a relation of type HasStaticValue. On the left, two DynamicDrillingSignals are storing downhole pressures. They are both connected to their corresponding semantical node of type DrillingData by HasDynamicValue relations. On the top is a DrillingData node, corresponding to the downhole instantaneous sensor reading (yellow). It does not have any associated signal, as it is not transferred to the real-time server. However, by representing it, one expresses the fact that the two available pressures have the same origin but correspond to two different processing.

In order to ease the manipulation of the semantical model, we chose to introduce specialization at the DrillingData level. To do so, we considered standard drilling signals are natural sub-types: Hookload, BitDepth, SPP, ROP, … However, these types are not unique as subtypes; an instance of the Hookload type can be a measured signal, a set-point, and also an uncertainty estimation. The rest of the semantical model, as described below, is used to perform that kind of distinctions.

Dimensionality of the data can be difficult to represent in an unambiguous way. In order to achieve sufficient generality, we consider drilling data as mappings from a space of dimension $\boldsymbol{d}+\mathbf{1}$ (the domain plus the time dimension) to a space of dimension $\boldsymbol{r}$ (the rank). Mathematically speaking, drilling data item can be represented as

$$\mathbb{R}\times\mathbb{R}^{d}\longmapsto\mathbb{R}^{r}.$$

A pressure is a single scalar value and is of rank 1 while a three-dimensional velocity is of rank 3. A collection of pressures along the annular has domain dimension 1, since one coordinate is necessary to locate it, a pump pressure has domain dimension 0 since no number is necessary. A collection of several three-dimensional velocities, measured at different depths and at different radial positions will have domain dimension 2, since in that

case 2 coordinates are necessary to locate it. This approach can also be used to describe rheology measurements (where both the domain -the shear rate- and the rank -the shear stress- are of dimension 1) or complex data.

The data dimensionality is expressed at the DrillingData level. Basic time aspect information is also expressed at this level, such as the expected refresh rate of the associated signal (if any), or a single Boolean whose value indicates the validity of the associated signal. This field has been included to easily manually notify consumer application that a given signal should not be trusted / applied anymore.

Unit and quantities

Unit management is often a source of human errors, because of the constant care needed for potential conversions. We adopt the strategy to internally store all of our values using the SI unit system, which provides a standard and practical system.

However, it is still necessary, for each data item, to be able to specify the underlying quantity and SI unit. To do so, we first define the Quantity type, containing (integer) fields representing the dimension of each of the base units: mass, length, time, electric current, thermodynamic temperature, amount of substance and luminous intensity. All physical quantities are derived from these: a velocity has for example length exponent 1 and time exponent -1, a pressure length exponent -1, mass exponent 1 and time exponent -2.

Such a characterization is not enough for the proper manipulation of the various numbers involved in drilling operations. One of the main advantages of quantity management is the ability to perform meaningful comparisons: any numerical comparison requires some accuracy, depending heavily on the nature of the measurement in consideration. It turns out that our previous characterization does not provide all the required information for this specific purpose. For example, pressures can be used in very different contexts in drilling: pump pressures, formation strengths and mud shear stress are all different by several orders of magnitude, and one cannot use one single comparison accuracy, even if they all are pressures. We therefore introduce another type, called MeasurableQuantity. Each instance of this type refers to a Quantity, but also has an additional field corresponding to the meaningful precision used for comparisons. This is illustrated in Figure 24.

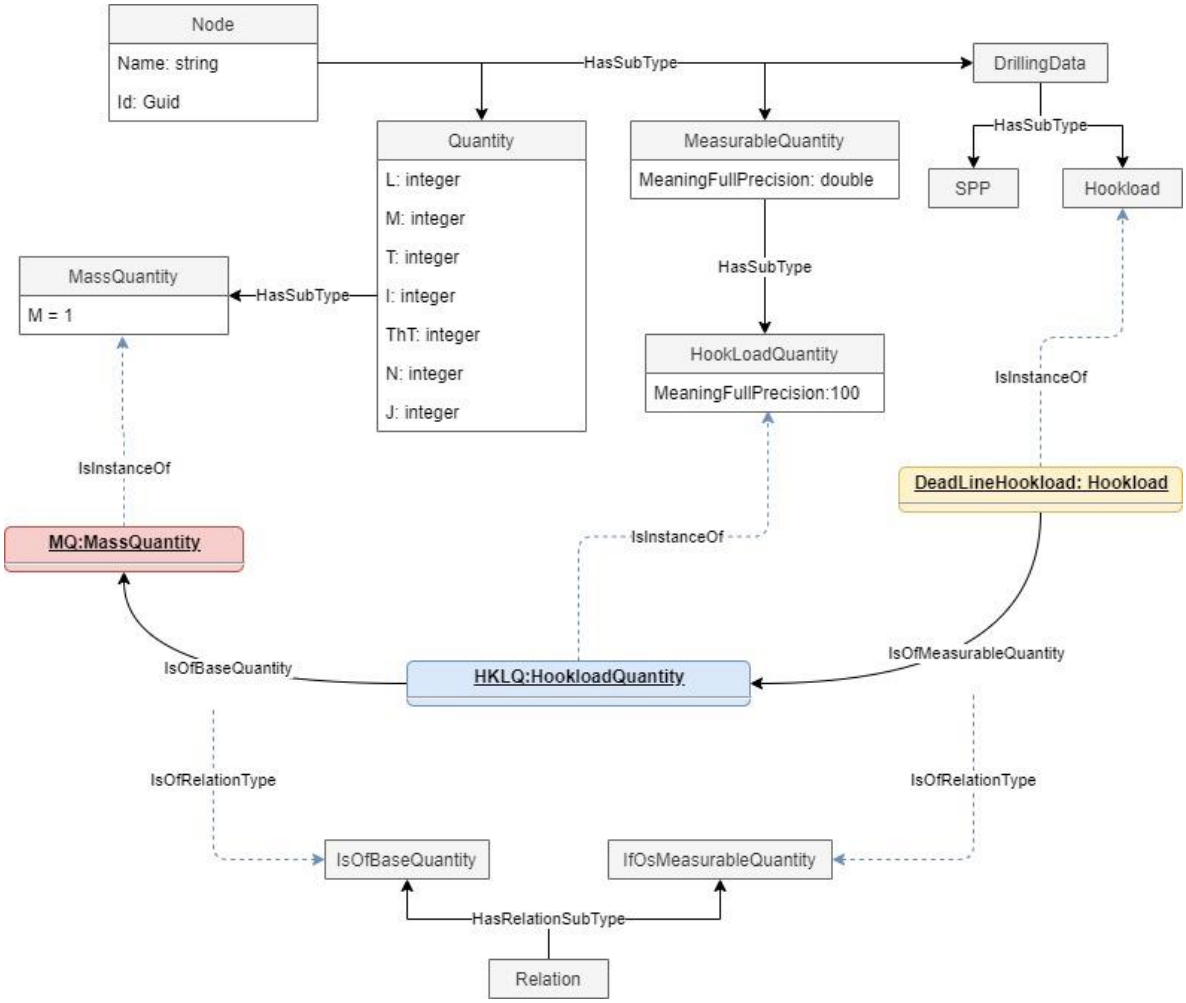

Figure 24: full representation (including types and relation definitions, as well as instantiation relations) of the management of units and quantities. Here, the semantical graph contains three instances: an instance of the Hookload type, linked to an instance of type HookloadQuantity. This hook-load quantity itself refers to a mass quantity.

One can then extend this formalism to include units: each unit is then defined by its conversion coefficients (coefficients a and b for a conversion $ax + b$) with respect to the default SI unit. One can further define unit systems (such as metric, US, imperial) that are collections of associations between measurable quantities and units. For example, the metric unit system contains the association between the fluid pressure measurable quantity and the bar unit for pressures.

Uncertainties

When analyzing measurements, it is important to account for the uncertainty attached to the sensor. In a similar fashion, computed values are uncertain: modelling assumptions, numerical errors, integration of measurements for filtering or model parameter estimation

- all of these contribute to computation uncertainty. Note however that not all signals have uncertainty: an example being exact set-points as desired values. The same applies to control commands or limits / constraints.

The uncertainties can take different forms. Sensor values are typically described by their accuracy, i.e. an estimation of the possible systematic bias and their precision, i.e. an indication of the reproducibility of the measurement. Uncertainties of computed values are often reported as standard deviations (therefore assuming normality), but most advanced uncertainty estimation techniques can result in arbitrary distributions, best described by histograms.

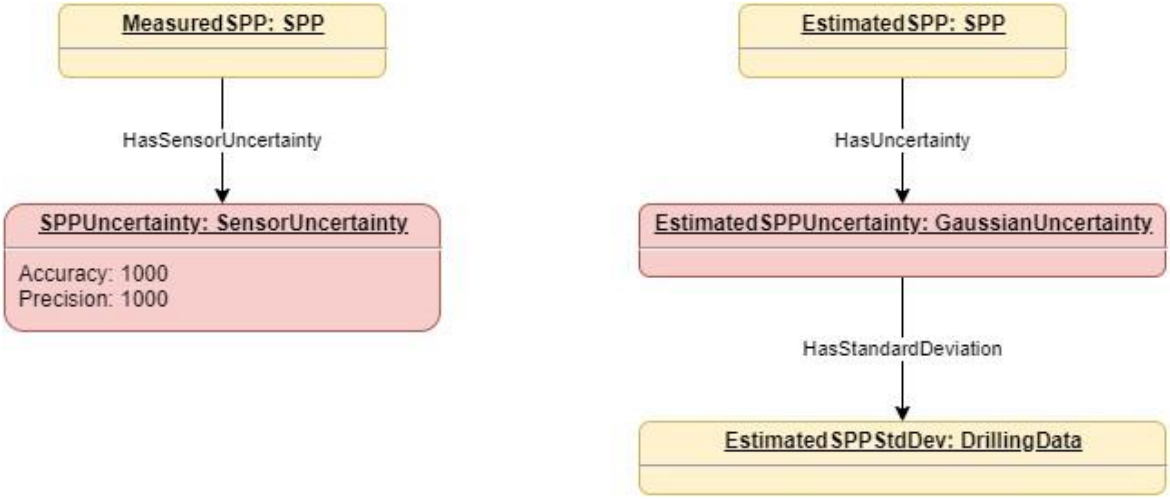

Figure 25: Example of some uncertainty description. The measured SPP is associated with the sensor's uncertainty, whose characteristics are fixed. This contrasts with the uncertainty associated to the estimated SPP (as computed by some numerical simulation process for example). It is noted as a gaussian uncertainty, but the associated standard distribution is itself a drilling data item, which may vary with drilling parameters and conditions.

Data validity

It is important to inform when a signal is valid or not. A signal may be invalid simply because of a hardware or communication error. For instance, if a sensor utilizes a 4 to 20mA current loop to transfer an analog measurement, a current smaller than 4mA is automatically categorized as faulty. Likewise, smart sensors have their own diagnostic capabilities and can maintain the validity of the associated signal.

But the validity of a signal may be conditioned by the current context. For instance, the validity of signals that are transmitted with mud pulse telemetry require a high enough circulation rate for the MWD pulser to work properly. Another example is related to hook-

loads or torques measured with an instrumented iBOP or saver-sub. In this case, measurements have a meaning only when the top-drive shaft is connected to the top of the drill-string.

It should be noted that validity concerns any type of signals and not only measurements. Indeed, set-points, commands, limits, estimations and even parameters can be invalid for arbitrary periods of time. For instance, an estimated signal may be invalid because the last calculation of that signal has failed, or a set-point signal that is transmitted by the DCS is invalid because of a loss of communication with one of the programmable logic controller (PLC) in the DCS network.

Subsequently, the signal class has the additional property “Valid” that takes Boolean values. This property is maintained by the originator of the signal.

We may further wish to characterize the generic conditions that ensure the validity of some given signal. We therefore introduce another class hierarchy, namely the ValidityCondition type. This one is subtyped into specific validity predicates, such as GreaterThan, LesserOrEqual, Different, Equal that encode simple signal comparisons. Together with logical operation nodes (And, Or, Negate), advanced validity conditions can be expressed. In Figure 26, the fact that a downhole signal transmitted via mud-pulse telemetry requires sufficient flow-rate expressed using a simple GreaterThan (GT) condition, that performs the comparison between the current flow-rate in and a static minimum flow-rate parameter.

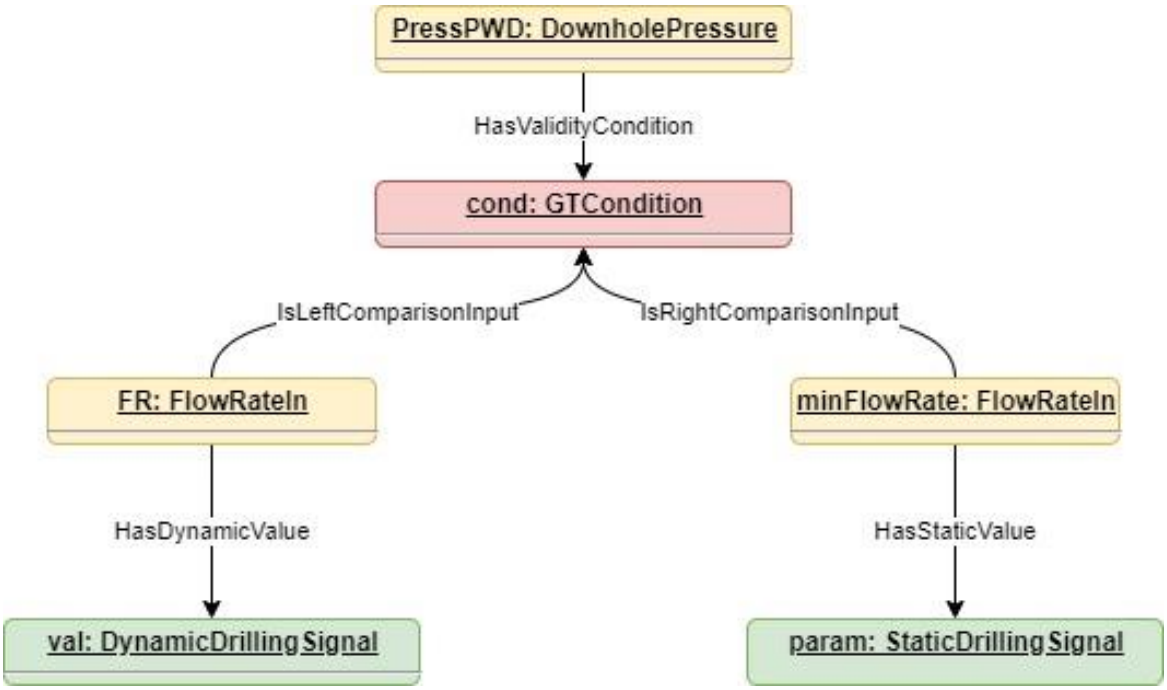

Figure 26: Example of a simple validity condition.

Time Management

Most of the information required to correctly interpret the time aspect of the real-time signals is included in the DrillingSignal type (which has two time stamps: acquisition and source) and in the DataFlow folder (see the corresponding chapter below) where delays associated to transmission and processing (such as buffering for averaging) are expressed.

The Time Management folder contains indeed mostly information related to the (possible) synchronization of signals. It could almost belong to the DataFlow category, but we nevertheless chose to treat it separately. As will be illustrated by the specific OPC-UA implementation of a DDHub server (see chapter 10), synchronization can be an important feature of a real-time data management system. In many cases, especially when considering down-hole signals, there can be a quite considerable delay between the original measurement time and the time the signal arrives in the real-time server. This delay is caused by transmissions and processing, and can be characterized by the information mentioned above, but in order to remove inconsistencies between various signals it is tempting to perform synchronization. Synchronization is nothing more than the coordinated resampling of several signals, where the delay involved in the resampling depends on each signal, such that the resampled values all correspond to the same clock time.

This is what the time management main type, namely the SynchronizationGroup, expresses. Many “raw” drilling data items can be specified to be input to such a group (see Figure 27): this reflects the fact that these signals will later on be synchronized.

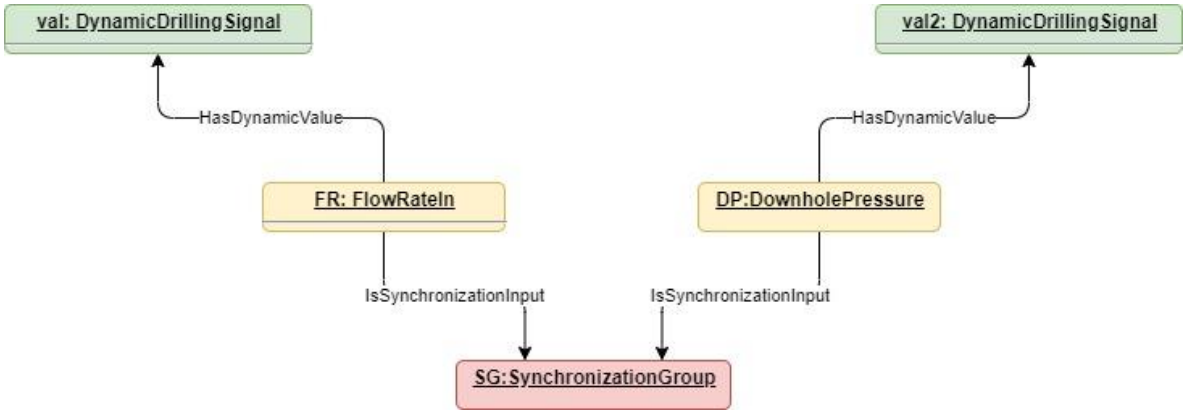

Figure 27: Instantiation of a synchronization group. Here, only the fact that both the flow-rate FR and the downhole pressure DP will be synchronized. The resulting signals have

not yet been created, nor the sub-component of the synchronization process (namely the time-based resampling).

The signals that result from the synchronization process should be clearly related to their source: this is done by specifying that they are the output of a resampling which itself has the raw signal as input; it should also be clearly stated that the synchronized signals do indeed share the same time stamps. This is indicated by the fact that they all refer to the same synchronization group. All those relations are illustrated in Figure 28.

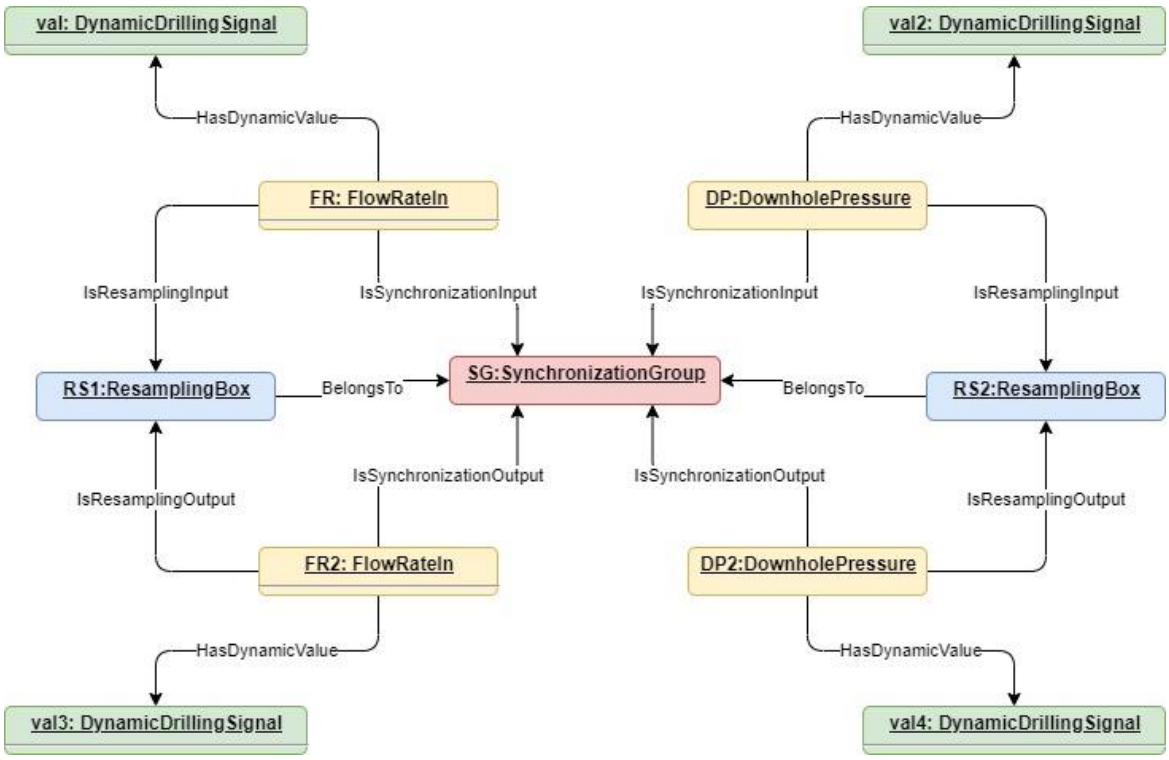

Figure 28: Synchronized signals. From this semantical sub-graph, a user can deduce that the flow-rate FR2 and the downhole pressure DP2 are deduced by resampling from the original flow-rate FR and downhole pressure DP. In addition, the resampling is such that both resulting signals are synchronized.

By introducing the semantic of the synchronization process, it becomes possible for any consumer to identify synchronized signals. Should an automation system require a set of such signals that already exist on the real-time server, it would not need to perform the synchronization itself or to ask the DDHub server to repeat the synchronization work.

Data Flow

Signals available on a rig are seldom raw measurements. There can be several layers of processing, transmission, and duplication between the first acquisition of the value of interest and its final arrival in the real-time data management system.

We choose to represent the transfer of information between data by data flow diagrams. We introduce a new node type, DataFlowNode which represents any processing unit. In turn, this type is subtyped to expose the different possible operations: we distinguish for example between acquisition from sensor value and classical filtering, direct transformations (that operate on single values, as for example conversions from dead-line tension to dead-line hook-load) and time base operations (such as derivations, integrations) that operate on data buffers. Each DrillingData node is the output of at most one DataFlowNode (it has therefore one single origin), but can be input to several DataFlowNodes (it can therefore have many "functions", or roles).

This formalism can be used in many ways. The most obvious is the necessary description of possible statistical treatment, of crucial value for downhole values. In such a case, statistical treatment information is mandatory for correct interpretation of the signal. But the data flow diagram can also carry relevant information about the delays inherent in data transmission. Getting access to transmission delays is mandatory for post-synchronization of signals, which is itself a requirement for detailed analysis of downhole conditions. Also, by expressing a signal's function, as input to a processing node, one can easily identify the role of some signals in the drilling process: set-points are recognized as inputs to control loops, commands as outputs of such items and inputs to controllers. It is also possible to specify that a measurement is used as control value for the same loop.

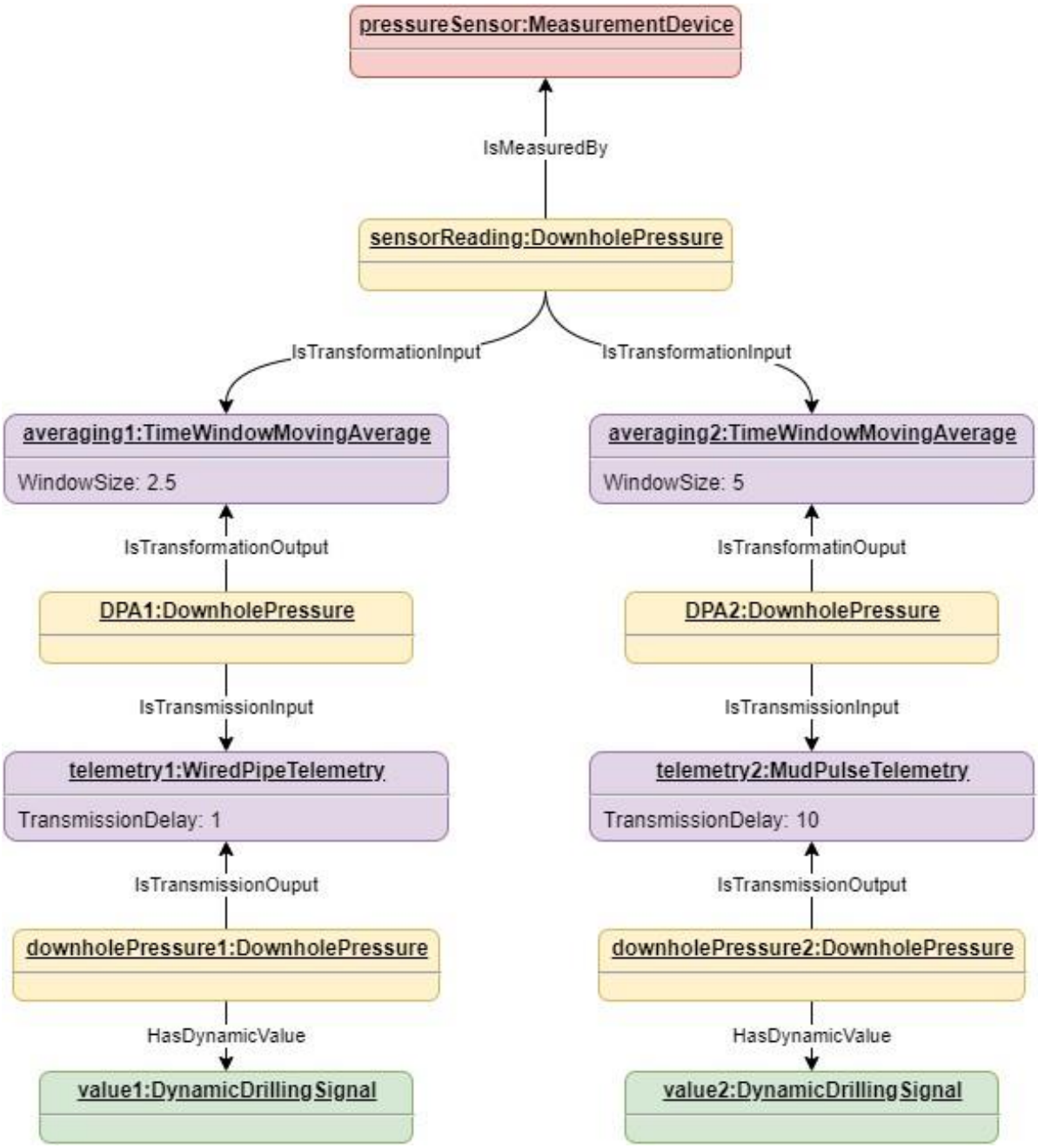

Figure 29: Data flow diagram for the processing and transmission of downhole. There are two downhole signals

Physical Locations

Every signal is physically located somewhere: a measurement has a natural location - the one of its associated sensor. A modelled value is also naturally attached to a geographical

point. The physical location of a pump-rate set-point is defined as the location of the corresponding mud pump, and so on…

A proper explanation of signal description type is in some circumstances necessary. This is achieved in the DDHub by using the following types and relations:

- *HasPhysicalLocation*: a relation type whose domain is the Signal type, and range the Location type.
- *Location*: a node type that represents a physical location. It is linked to a ReferenceFrame, another location representing the frame's origin and another Signal that represents the coordinates. The relations to be used are respectively *HasReferenceFrame*, *HasReferenceFrameOrigin* and *HasCoordinates*.
- *ReferenceFrame*: the type that represents a system of coordinates. It is first sub-typed along the dimension (one, two, three or four) of the coordinate system, its nature (cartesian, spherical, curvilinear…) and finally drilling specific specializations.

By using this approach, one can recursively refer to several coordinates systems: for example, a PWD is first defined by is distance to the bit, along the upward-positive oriented one-dimensional curvilinear frame following the wellbore centerline. The origin of this reference system is the bit location, itself defined with respect to the drill-floor (another location that serves as an origin in our case) by the bit depth signal (the coordinates) in a positive downward curvilinear frame (see Figure 30).

The variety of reference frames, and the flexibility attained by allowing references between frames allows unambiguous representation of the different type of positioning involved during drilling operations.

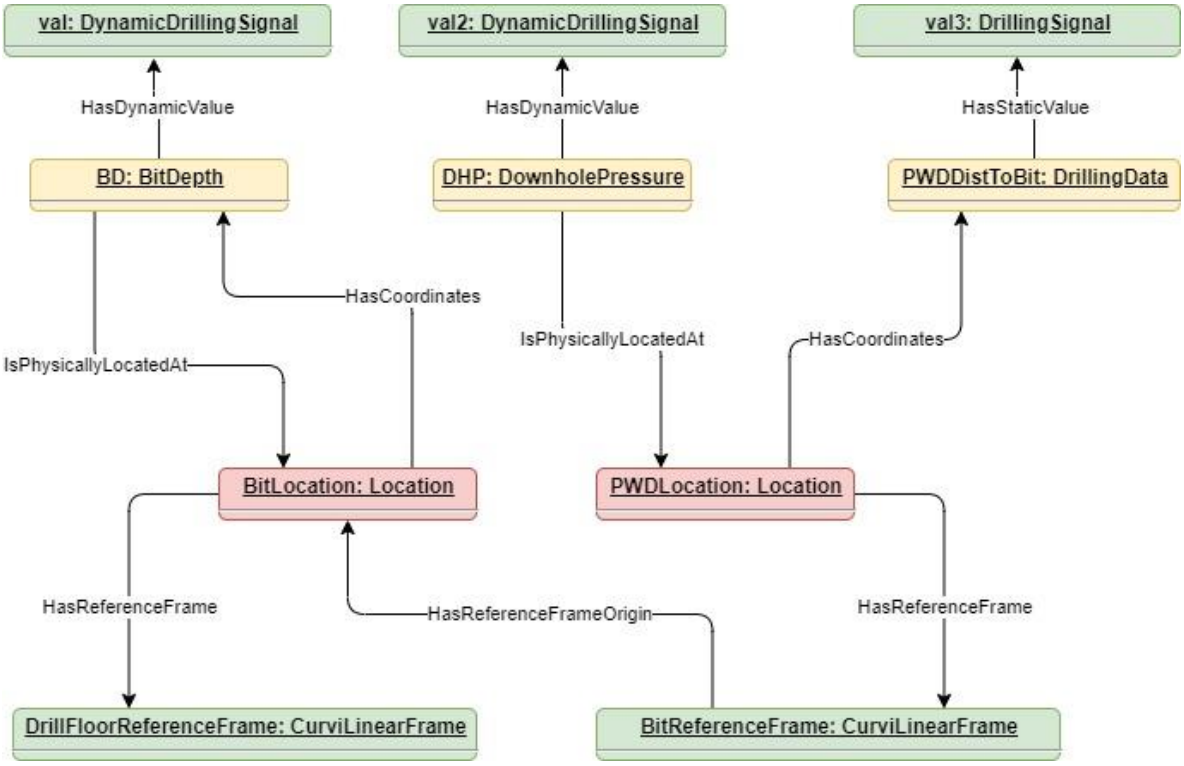

Figure 30: Example of combined locations. The downhole pressure data (DHP) is located at the "PWD Location". This location has coordinates "PWDDistToBit" (in our case a static configuration parameter) in the curvilinear reference frame whose origin is the bit location. The bit location itself has coordinates "BD" (this time the drilling data is dynamic and can change at any second) in the curvilinear reference frame whose origin is the drill-floor.

Hydraulics

In the previous paragraph, the types and relations necessary to describe the physical location of drilling signals were presented. When analyzing the drilling process, alternative views can be necessary. Some signals have a specific role in the mechanical process associated to the drilling activity: a deep understanding of the nature of some measurement or calculation with respect to the mechanical forces or pressure is key to several types of automation systems. The current sub-chapter and the next are devoted to the two main "abstract" physical systems (where the term physical refers to physics as a science) classically considered during drilling, the hydraulic and mechanical systems.

Proper characterization of a signal with respect to the wellbore's hydraulic system is achieved by linking the DrillingData instances to instances of this hydraulic system via relations of type IsHydraulicallyLocatedAt (this relation type is sub-typed into specialized relations such as IsPressureAt, IsVolumetricFlowRateAt, …).

The major difficulty is in the representation of the hydraulic system itself. Here, we chose a very generic approach, inspired from the meshing process involved during the discretization

of a wellbore for setting-up numerical hydraulic computations. We consider three main types of hydraulic elements:

- Hydraulic branches: those elements correspond to finite volumes that can contain the considered fluid. The geometry of the branch is expressed by sub-typing: Annular, OpenChannel and Pipe are the main sub-types. We made it possible to consider multi-resolution branches: a branch, for example the interior of the drill-string, can be decomposed into several branches, where each volume corresponds to a single drill-pipe or BHA component. A branch can in addition have its own reference frame, as defined previously.
- Hydraulic junctions: those elements connect two branches together. They can be located at any place in a branch: their locations within their left and right branches are expressed using the physical location mechanism described in the previous section. More specifically, the branches' reference frames are used there.
- Network: this is a collection of interconnected branches. One can then consider the top-side network as a whole, or the mud-mixing network…

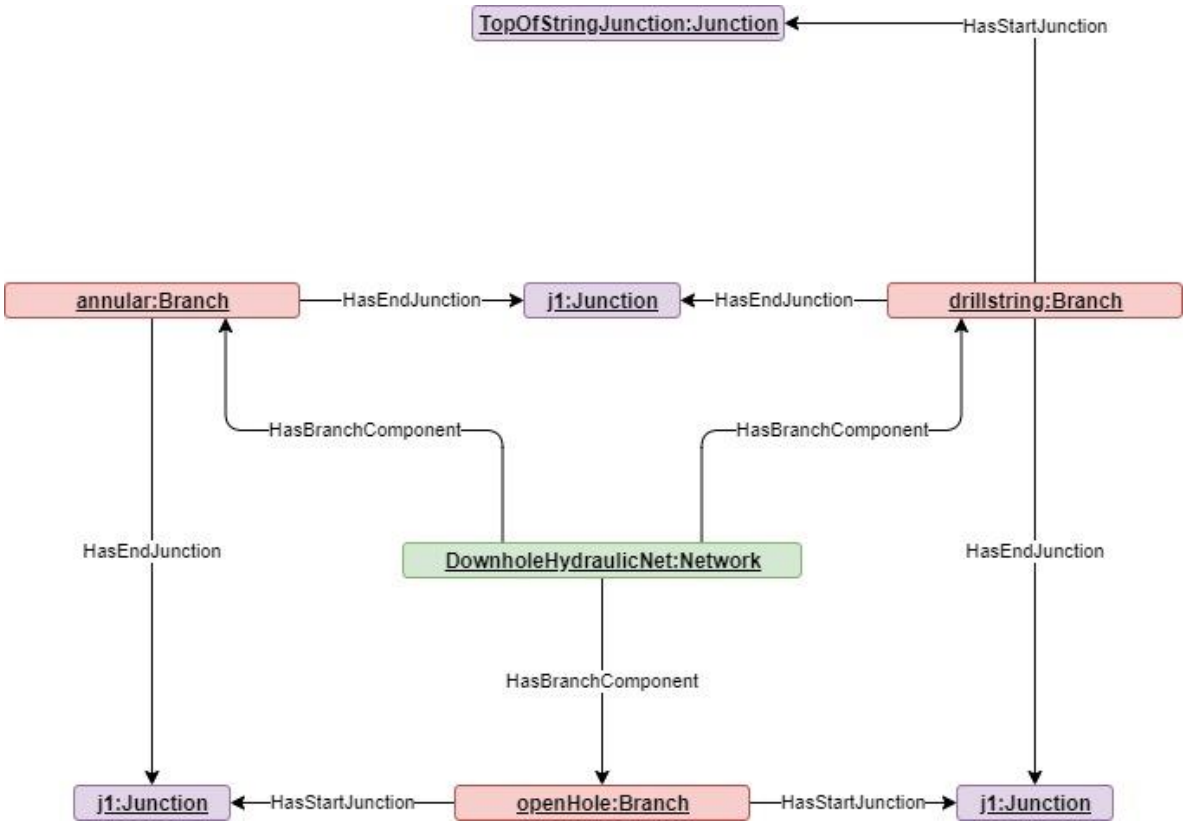

Figure 31: a simple hydraulic network, that describes the fact that the downhole network is comprised of three branches: the drilling string (interior), the open-hole and the drill-

string annular. The three junctions correspond to the same physical location (the bit), but are represented as three separate entities.

Both branches and junctions can have an associated state: for example, valves can be closed or open. The state value itself can be another real-time signal, either static (it will therefore be manually updated) or dynamic (when automatically operated).

Mechanics

The mechanical system is represented using the same mechanism as the hydraulic system.

Drilling Equipment

Finally, the positioning systems (physical, hydraulic and mechanical) all may refer to real equipment (for example the bit, top-drive hook,… can be seen as mechanical and hydraulic components and as physical locations). It is therefore necessary to include those in our model. We chose to only include them separately (and therefore not duplicate them depending on the considered role), and use the class equivalence principles here: the type Bit is then marked as being valid as an hydraulic junction and as a location. The same holds for most of the drilling equipment. The current list of drilling equipment present in our vocabulary is limited to those elements that were required for the different use-case implementations. The reason for this is that there exist already several good quality models that cover those aspects: the WitsML data model for example provides most of the required items. In addition, many companies, and sometimes many installations within the same companies, develop their own models. It is then natural to use the integration capabilities of semantical modelling to include external models within the DDHub system.

10. Discovery and queries

The semantical description of drilling signals sketched above can only be useful if automated treatment of the information is possible. Then drilling automation systems can automatically discover and identify signals that correspond to their needs.

Automated analysis of a DDHub graph can be done in (at least) two ways:

- Browsing: this is the conceptually simplest approach. It consists in local exploration of the DDHub graph to gain information about the available signals. For example, given a specific Hookload, browsing can be used to first inspect the location of the associated measurement, such that dead-line hook-loads can be distinguished from top-drive ones. This approach is suitable for fine analysis of specific signals.
- Querying: this means constructing some specification that some nodes must satisfy, and algorithmically retrieving all the nodes present in the graph that satisfy the criteria. This is similar to classical data-base querying (as provided by the SQL language) and is a powerful tool that enables automatic identification of relevant signals.

The remainder of this chapter is devoted to the querying system that has been developed during this project. It is classically based on predicate first order logic and is fully compatible with the standard knowledge extraction languages such as SPARQL used in semantical engineering.

Base rules and concepts

The querying system relies on two interconnected constructs, called Base Rule and Concept. A base rule serves as a logical predicate that characterizes local properties of a node within a graph. It is formally a Boolean function of the form

$$BaseRule\big(n, d, R, C, k, w, (P_i, v_i)\big) \in \{0,1\}$$

This functions counts the number of relations that have a given specification, and returns 1 if this number is as prescribed. More precisely, denote by $\#B$ the number of relations of type R from (if d is true) or to (otherwise) the node n, where the other end of the relation satisfies the concept C. Then, if w is true, the function returns 1 when $\#B = k$, and 0 otherwise. If w is false, the function returns 1 if $\#B \neq k$, and 0 otherwise.

In addition, it is required that for all the property-value pairs (P_i, v_i) the node n's value of the property P_i equals v_i. If this condition is not satisfied, 0 is returned.

It returns 0 (false) otherwise. A Concept is nothing more than a logical formula built on base rules, with an additional requirement on the node type. We typically represent the logical formula in conjunctive normal form (CNF), such that

$$Concept(n, T, CNF) = (n \text{ is of type } T) \wedge (CNF)$$

Where the operators ∧ and ∨ denote the logical AND and OR and the formula CNF can be expressed as

$$(BaseRule_0 \vee BaseRule_1 \vee \cdots \vee BaseRule_k) \wedge \cdots \\ \wedge (BaseRule_i \vee BaseRule_{i+1} \vee \cdots \vee BaseRule_l)$$

This simple construct allows the specification of advanced graph patterns, and because base rules can refer to other concepts, extend to global properties of the graph by recursion.

Consider for example the almost empty concept obtained by setting the type T to HookLoad: then, for a given DDHub graph the only nodes for which the concept function returns true are the HookLoad nodes. This can be seen as a simple type filtering. Consider now adding one single base rule with the following parameters:

- $d =$ true
- $R =$ IsMeasuredAt relation type
- C is the concept that filters nodes with type DeadLine
- $k = 1$
- $w =$ true

Then, the concept function will return true for the nodes of type hookload, that have exactly one relation of type IsMeasuredAt pointing to a node of type DeadLine. In other words, it will identify all the dead-line hook-loads. In practice, most of the queries needed to set-up a drilling automation system should be relatively simple, and consider only very local properties of the searched nodes. However, it is possible to construct concepts that express rather complex patterns: for example all the nodes that point to some hydraulic location located between the bit and the bell-nipple, all the nodes whose main origin is a measurement, and not n complex calculation… The latter case is motivated by some classical drilling signals, such as the downhole ECD. This quantity is intuitively a measurement, but in fact the “path” between the original measurement (a pressure reading at the PWD) and the final downhole ECD is rather complex, since it is based on some depth normalization, where the considered depth (in TVD) is itself deduced from rather advanced geometrical calculations. It is however possible to specify a concept that identifies the signals that have a connection to measurement that avoid complex calculations (see Figure 32).

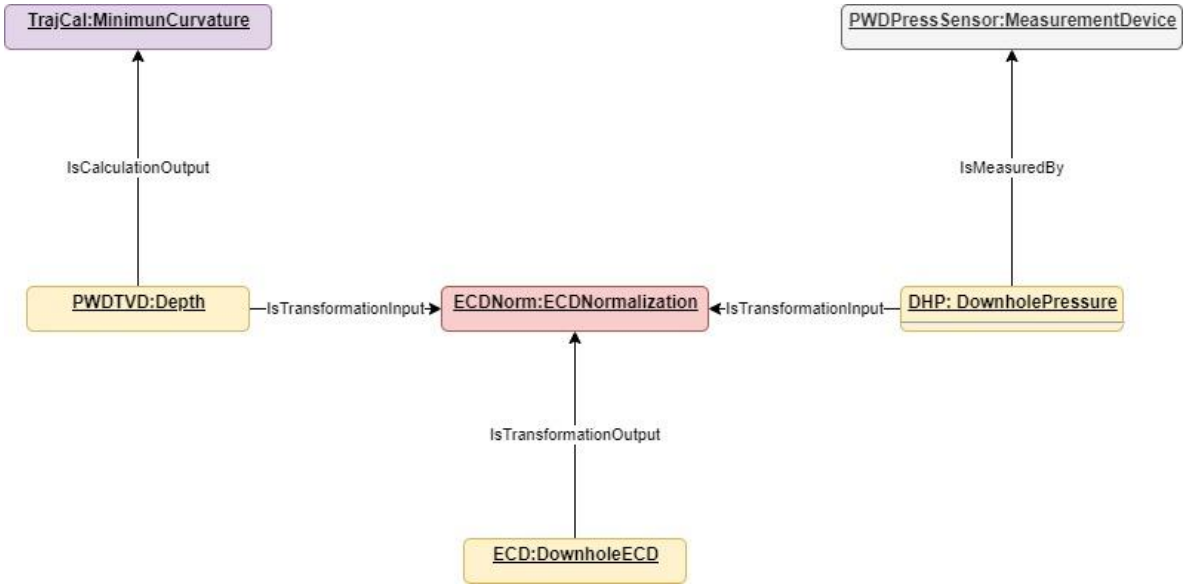

Figure 32: even if the downhole ECD is not a measurement, there is a path between the ECD node (lowest node) and the measurement device (top-right) that avoids complex calculations (the minimum curvature computations at the top-left) and only simple processing nodes (such as conversion or normalization).

Construction rules

The logical constructs defined above play another important role in the semantical modelling: they allow specifying some structures that nodes of a specific type should satisfy. This enables the validation of some DDHub graph, and the identification of some parts of the graph that are not valid. For example, it is possible to specify that all the nodes of type HookLoad should be linked to measurable quantities themselves referring to some Mass base quantity, or not linked to any measurable quantity, as illustrated in Figure 33.

Figure 33: two hook-loads are represented. The top-one is not valid since it is related to a PumpPressureQuantity, itself pointing to a PressureQuantity. The second one is valid.

It is possible to specify rules for every DDHub type that define the acceptable local configurations. This ultimately leads to a full unambiguous semantical definition of the main terms used on a drilling rig.

Although the mechanisms to handle type definition rules have been implemented and tested, most of the definition work still needs to be finalized, as it requires collegial definitions, and is a natural part of the future phases of the project.

11. Implementations

The implementation can be split in several components (see the first definitions in chapter 7 and Figure 20):

- The DDHub server itself, that stores the semantical graph.
- The DDHub source accessor interface, that allows access to some Real-time server.
- The DDHub adaptor interface: that allows access to the DDHub server.

Regarding the first part, any modelling tool that allows the definition of types, and the construction of graphs can be used. We have so far considered two implementations: a C# one and an OPC-UA one. The different interfaces have all been written in C#.

Note that all the code is open source and available in an open Git-Hub repository.

C#

The first implementation of the DDHub structures has been done in C#, an object oriented language which is part of the .Net family of languages. This language allows a direct translation of the different parts of the vocabulary: indeed, the class construction has been automated, so that from an xml specification of the vocabulary, the corresponding classes are automatically generated. Note that specific care has to be taken when considering the DrillingSignal nodes: they correspond typically to signals present in the Real-Time server itself. Apart from the specific situation where the DDHub server and the real-time server are hosted by the same infrastructure, they therefore won't be represented in the DDHub graph. We use instead EndPoint nodes, that store the addresses of the signals in the real-time server. The EndPoint type is generic, and can be subtyped into WitsMLEndPoint, OPCEndPoint, WitsEndPoint, where each subtype specializes the addressing information.

Some simple graph structures have then been implemented to represent a DDHub graph, together with the classical graph exploration algorithms.

Finally, the graph implementation includes query evaluation: the queries themselves can be represented in an xml format (which eases their communication) and simple algorithms have been implemented that returns, given a graph and a query, the nodes from the graph that satisfy the query.

As stated above, the different interfaces have also been implemented in C#: they enable access to the different components of a real-time data system.

OPC-UA

A dedicated OPC-UA implementation was performed. Because this technology enables both the storage of typed data, semantical relations and real-time values, it is natural to consider fully integrated solutions where both the real-time server and the DDHub server coexist. We review a few details of the implementation here.

Model implementation and basic interfacing

The editing of the DDHub graph is performed via server methods: dedicated methods for adding, removing, and editing nodes and relations have been implemented. Those methods are then called by client applications (typically via the C# interface), where the possible parameters are passed using JSON or xml serialization.

Loading of the DDHub graph is also achieved via specific methods. The graph itself is returned via xml serialization. In Figure 34 the different methods implemented at the server level are shown.

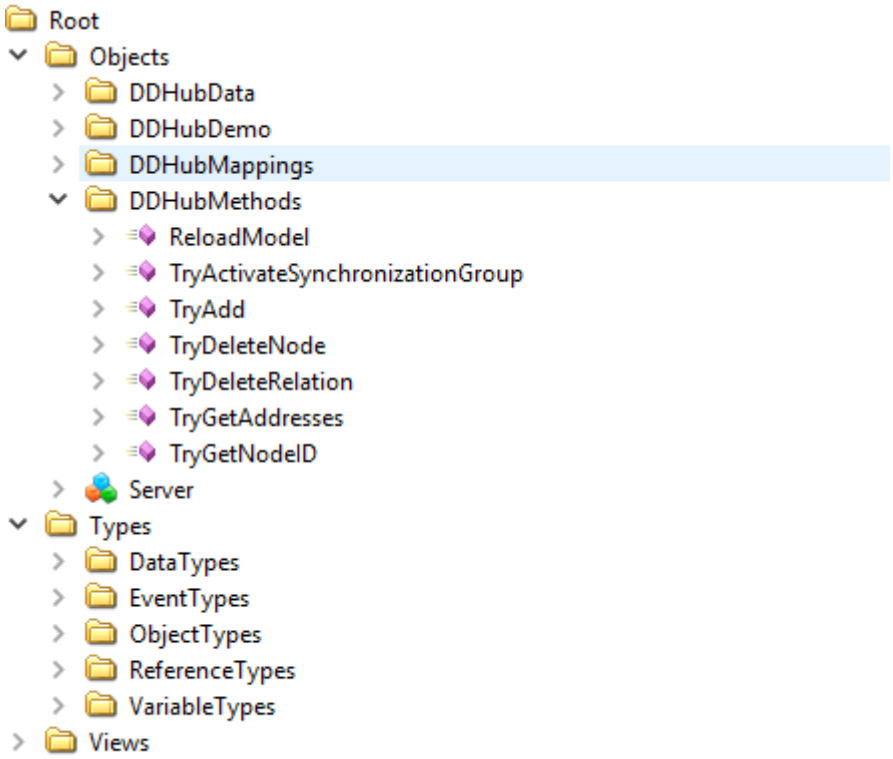

Figure 34: the different DDHub methods implemented at the server level of an OPC-UA server. The picture is taken from the UA-Expert tool from Unified Automation.

The data model itself is almost directly transposed into OPC-UA types (see Figure 35). The elements that are not obtained by direct translation will be covered below. In order to fully specify the OPC-UA translation of the DDHub model, we chose to expose the different mappings between the DDHub node types and their OPC-UA counterparts (see Figure 36). By doing so, alternative implementations can be considered (such as ones that re-use existing implementations), but a consumer application (or the implemented interfaces) will still be able to translate the OPC-UA model into a DDHub one. Note that a similar exposure of the implementation details is performed at the fields level: we chose here to use a specific relation type (HasDDHubProperty), which inherits from the standard OPC-UA HasProperty relation (see Figure 37), to link instances to their DDHub fields. Here as well, a consumer can automatically convert the OPC-UA property to its DDHub counterpart.

In agreement with the OPC-UA common practice, we chose to display the various instances of DDHub nodes in a specific folder, itself root of a folder sub-structure that reflects the DDHub inheritance folder. Each instance is then displayed within the folder that corresponds to its type. This approach eases manual exploration of a server: it is not mandatory for automated explorations, but can be useful in the adoption phases of the OPC-UA implementation.

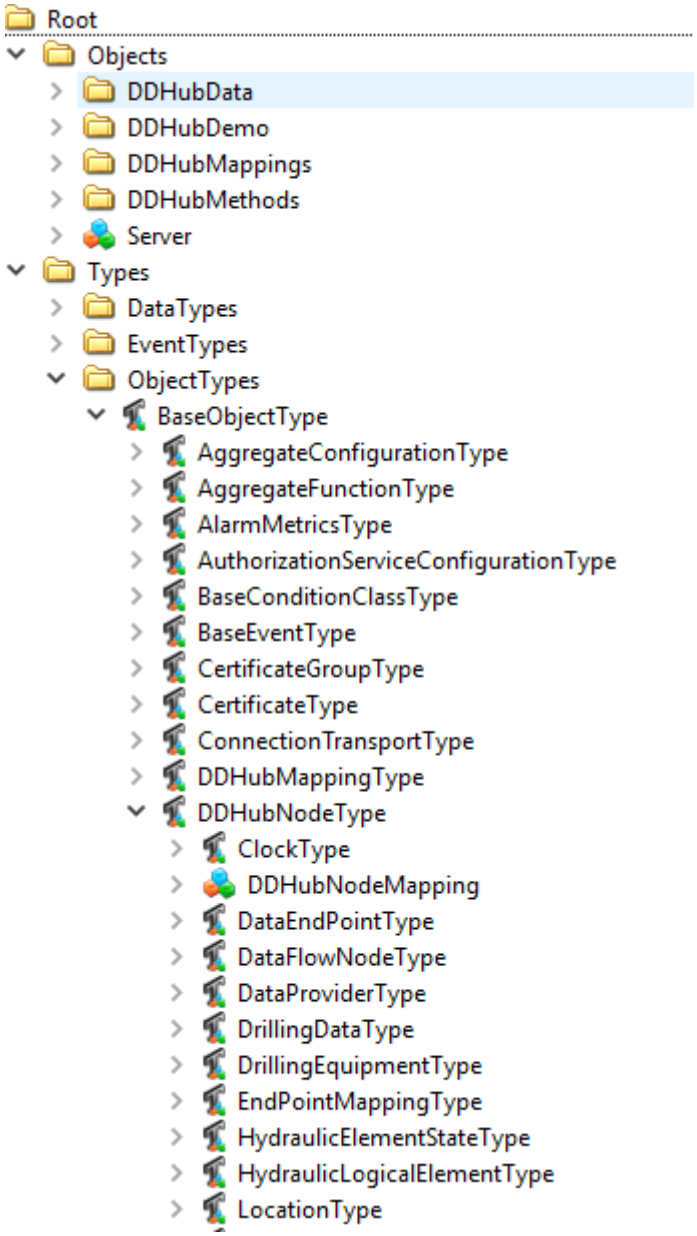

Figure 35: the major part of the DDHub model (at least the node types) is directly converted to OPC-UA types, as can be seen here: the DDHubNode type is integrated into the OPC-UA types by inheritance from the BaseObjectType type. From there, almost all the DDHub types (apart from the DrillingSignal types) are integrated by OPC-UA inheritance.

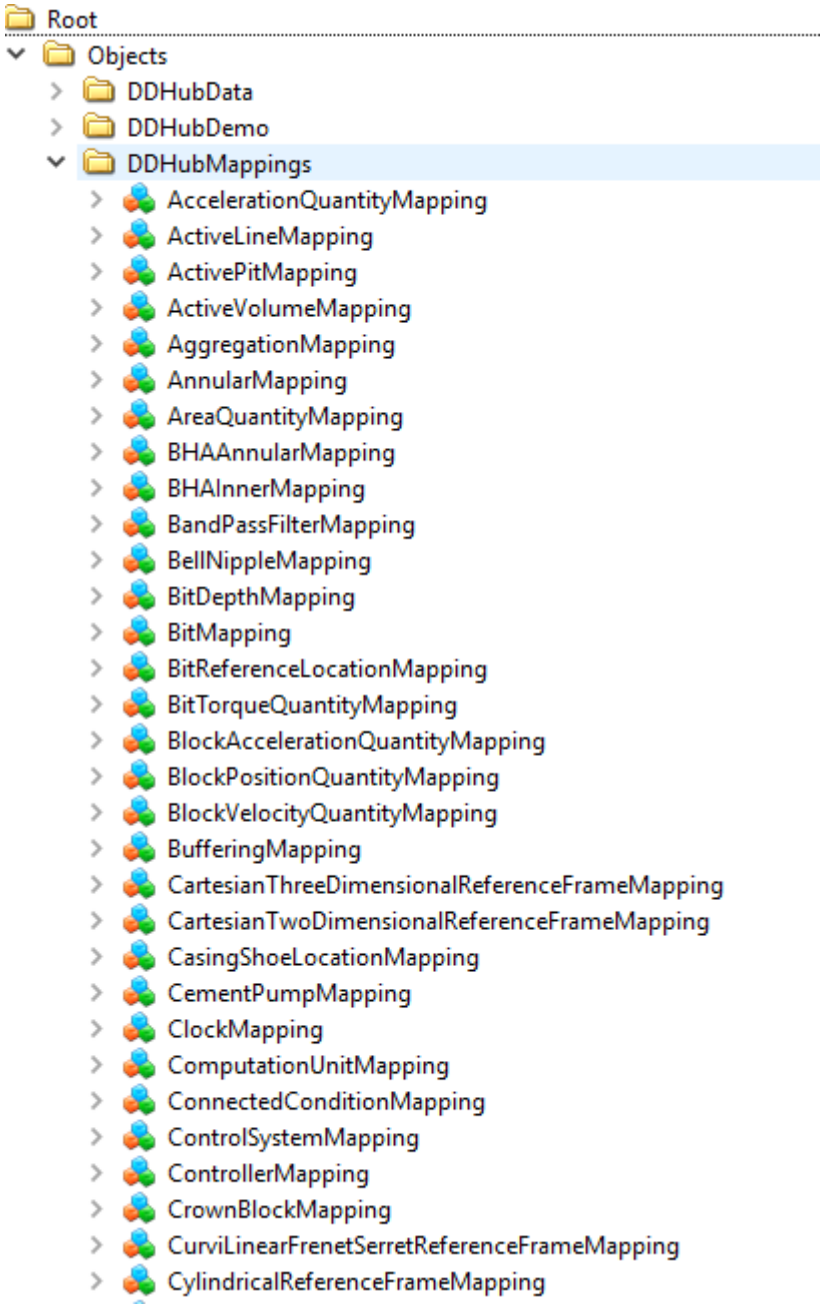

Figure 36: the OPC-UA implementation mappings. Each of the shown mappings has two fields: one that points towards the DDHub original type (with a string identifier for the

type) and one that points to the corresponding OPC-UA type. By doing so, any OPC-UA object can be translated into its DDHub equivalent.

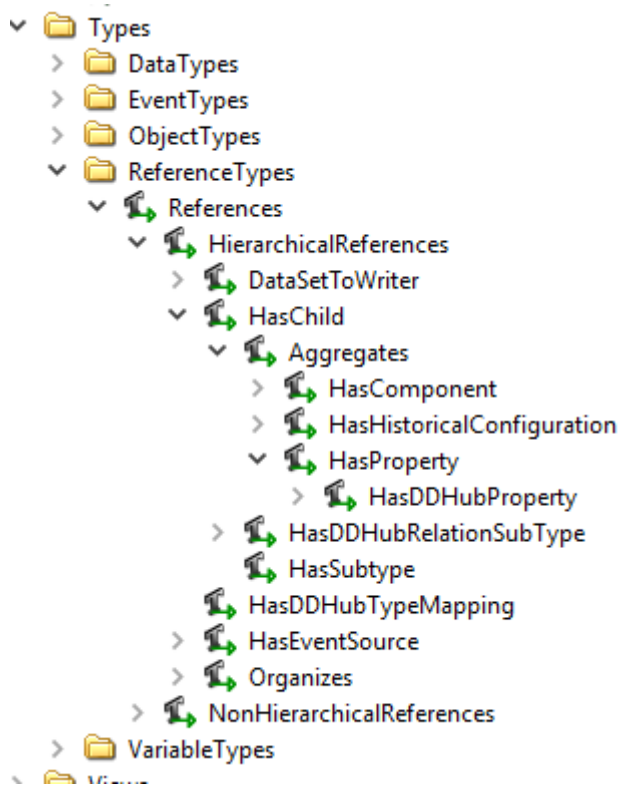

Figure 37: the HasDDHubProperty relation extends the default HasProperty one, which is used to express the fields of various instances. A consumer application, when browsing the properties of a specific instance, can therefore know whether a given property reflects a DDHub one. Note also the presence of the HasDDHubRelationSubType reference, used

to express the DDHub inheritances between relations, as it can conflict with the standard OPC-UA reference inheritance relations.

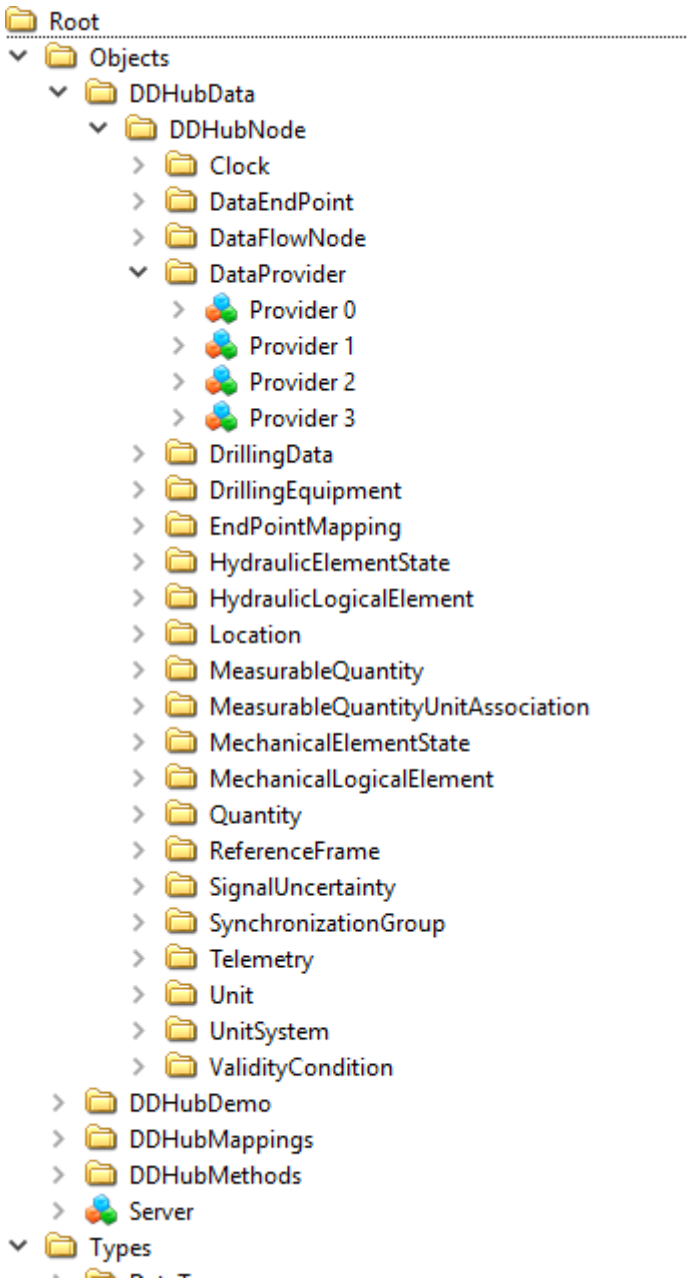

Figure 38: instances are stored in a specific folder, with a subfolder structure that reflects the DDHub inheritance one. This is the typical way of displaying information in the OPC-UA community, as it eases the manual exploration of a server.

The implementation of the DDHub relations is very similar: we use here the OPC-UA references that fulfill the same semantical function. The only restriction is that OPC-UA references have to inherit either from the Hierarchical or NonHierarchical reference types. It is therefore not possible to have a root DDHub relation type, as some of the specific sub-relations will naturally be of one type or the other. In order to circumvent the problem, we chose to introduce a specific DDHub reference inheritance relation. that will be used

between the reference type definitions to introduce the proper relation, possibly across the OPC-UA inheritance structures.

Real-time OPC-UA integration

As mentioned above, not all the parts of the DDHub model are directly translated into OPC-UA. The DrillingSignals (dynamic and static), since their vocation is to store the real-time data itself, are not translated, but we use instead the standard OPC-UA variable types (such as the standard AnalogItemType). The original DDHub relations HasSignal, HasStaticSignal and HasDynamicSignal still exist, but now relate a “pure” DDHub node (of course of type DrillingData) to a native OPC-UA one. Special care is taken during the possible instantiation of signals via the dedicated interfaces to select the most adequate OPC-UA type (we use there mostly the rank and domain information, as well as the data type information available from the DrillingData object). Since the OPC-UA variables already have the notions of server time-stamp and acquisition time-stamp, we use those to express the DDHub equivalent.

This pragmatic approach has several advantages. In particular, it allows to treat the different signals in a pure OPC-UA fashion. This enables several scenarios; for example, one can easily consider an existing OPC-UA signal, without any DDHub description, being integrated at a later stage to some DDHub graph. This is easily done by the introduction of the proper reference in the graph. This realistic scenario (a service company provides a signal without semantical information, and some third party provides the relevant information later) has been demonstrated during the latest demonstration (see chapter 12). It also allows the signals themselves to be accessed through completely standard opc communication: there is no need for dedicated interface for the classical read and write operations. This leads to some slightly different architecture, where most of the operations concerned only with real-time signals are performed using native technology, but the semantical aspects are still handled through a DDHub interface (see Figure 39).

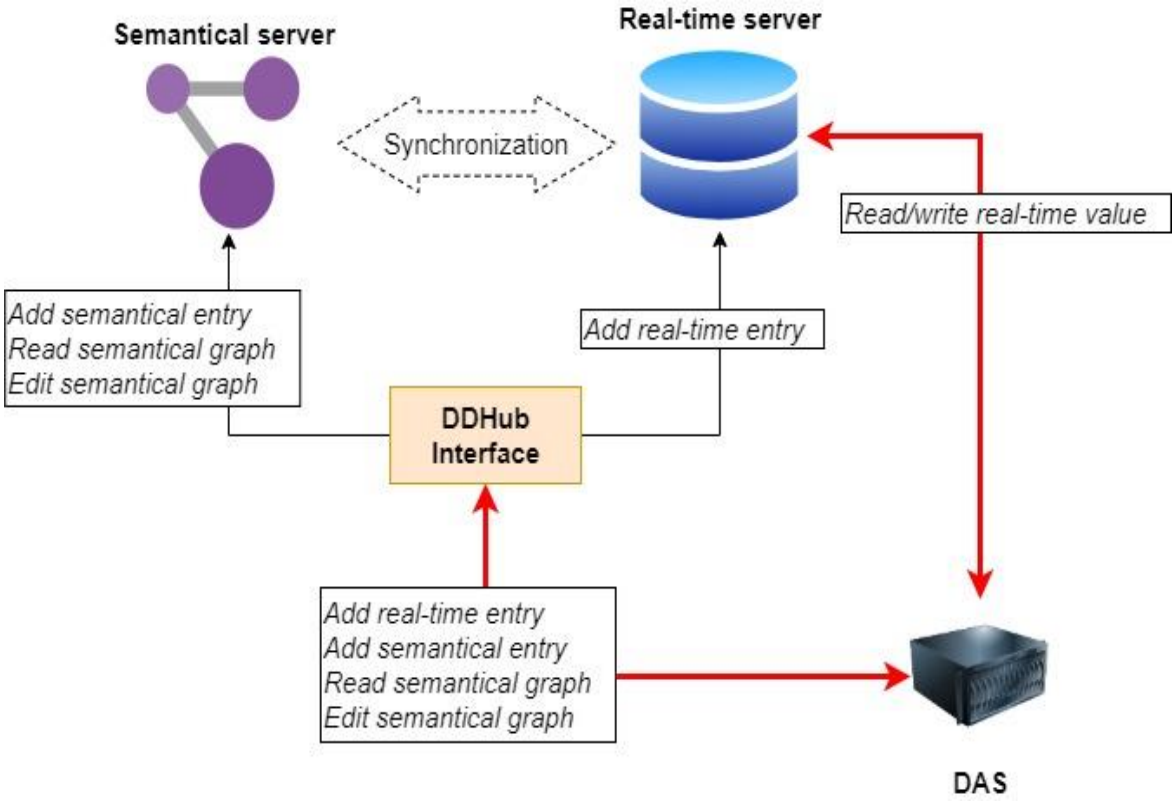

Figure 39: architecture resulting from an OPC-UA implementation. Direct access to the signals is possible in a classical way, such that only the access to the semantical information requires DDHub interfaces.

Synchronization interface

Finally, a special feature has been implemented in the OPC-UA case, namely signal synchronization. Synchronization is activated by consumer applications, using a server method (shown in Figure 34).

This method takes as argument the identifier of a synchronization group. Then, for all the signal belonging to the group, a synchronized version is instantiated. The semantics of the synchronization process are added to the DDHub graph (since the synchronized signals are nothing more than new signals, their meaning should be exposed as if they were “original” signals), and finally the newly created signals’ values are regularly updated by the server itself. It means that the server automatically creates the graph displayed in Figure 27 from the input graph of Figure 28.

Of course, the synchronization process requires configuration: refresh-rate, overall delay of the synchronized data (with respect to the server clock) and interpolation delay for each of the signals belonging to the group. The refresh-rate and overall delay are fields of the synchronization group and are therefore provided by the application that did instantiate it (it is possible to instantiate the group and activate it simultaneously with the implemented method). The individual delays for signal alignment are deduced from the graph properties:

when a signal has a source time stamp specified, this one is used. Otherwise, the server searches for delay (transmission and processing) information and uses it.

The synchronized signals are available to any consumer application, not only the one that triggered their creation.

Web API

As part of the implementation comes a DDHub Web API. The API is provided as REST http services with several route calls to facilitate the development of consumer applications. The main advantage of using the Web API is that the communication with the DDHub server becomes not only language independent but also easily implementable, since http communication is literally supported by every programming language available.

The Web API service does not store any data but is rather synchronized with the OPCUA server to mirror the DDHub model and all the changes that happen to that model, including real-time data changes. However, for latency reasons, we recommend streaming real-time data via native OPCUA capabilities. For that the API provides the OPCUA node ids of all DDHub nodes, so that a consumer application can address them directly via OPCUA communication.

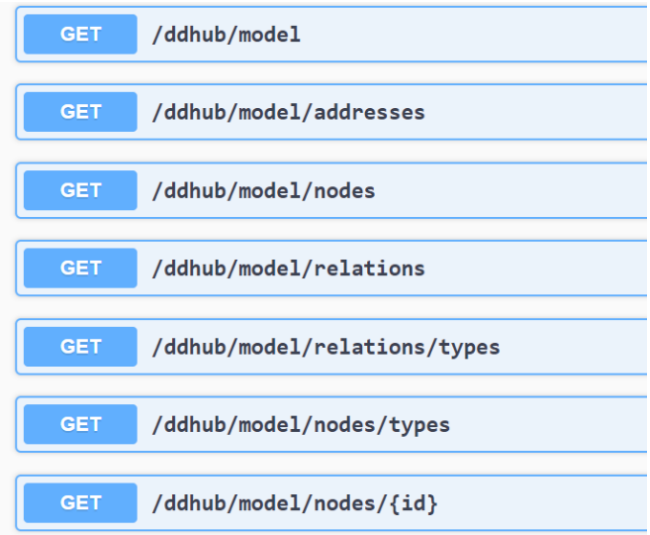

Figure 40. Basic DDHub model discovery routes

Figure 40 shows some routes that can be used to explore the DDHub model. Typically, one would use the route /ddhub/model to get the whole DDHub model in a JSON format or pick a specific node directly if the node ID is known. Calling /ddhub/model/addresses will return the OPCUA node mappings as mentioned earlier.

Figure 41 shows more a more advanced way to explore the DDHub model using DDHub queries. The Web API service as well as the c# DDHub reference implementation come with

a predefined set of the most common operational data queries. These can be provided via the route /ddhub/queries. Calling /ddhub/queries/call/{name of the query} will return all the nodes that satisfy the query. In general, queries can be used by consumer applications to check that the data is well described (remove ambiguities) and acquire the data as well. In addition to the pre-defined queries, a consumer application can define its own queries and apply them via the route /ddhub/queries/userQuery.

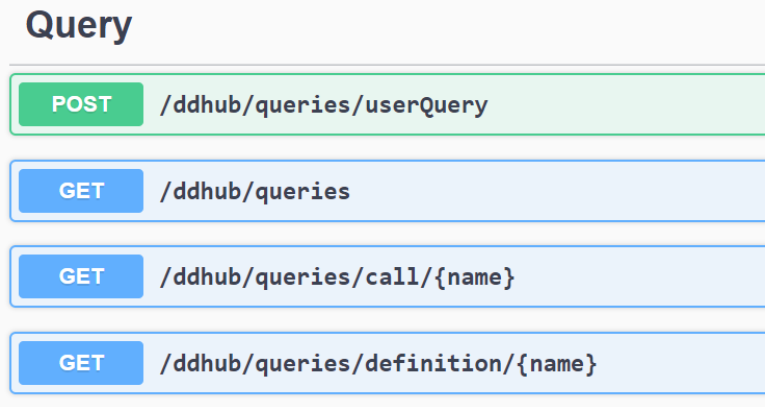

Figure 41. Routes for handling Queries

Edition and management tools

Several tools were implemented during the course of the project. They all targeted different stages in the development of the interoperability framework: semantical model development, instantiation and validation of "use-case" DDHub graphs, monitoring of interoperability or integration of sensors in a DDHub context. All these software tools are available via the dedicated Git-Hub), although some of them are still in an early prototype state, since they have mostly been used internally and did not require the same care as commercial applications. These applications are briefly described below.

Semantic editor

The development of the semantical model itself can be done manually (via manual editing of the various representations) but it quickly becomes a task difficult to maintain. We therefore chose to implement a simple editor, dedicated to the development and maintenance of our model (see Figure 42). Had we chosen to use Semantic Web technologies, and in particular OWL ontologies, the corresponding tool would have been the ontology editor Protégé[6].

With the semantic editor tool, it is possible to:

[6] https://protege.stanford.edu/

- Add, remove types for nodes and relations
- Add, remove fields for node types
- Move items in the inheritance structure
- Specify default attributes values for node types
- Export the model in:
    - XML
    - C# classes
    - OWL2 ontology
    - OPC-UA Node-set file

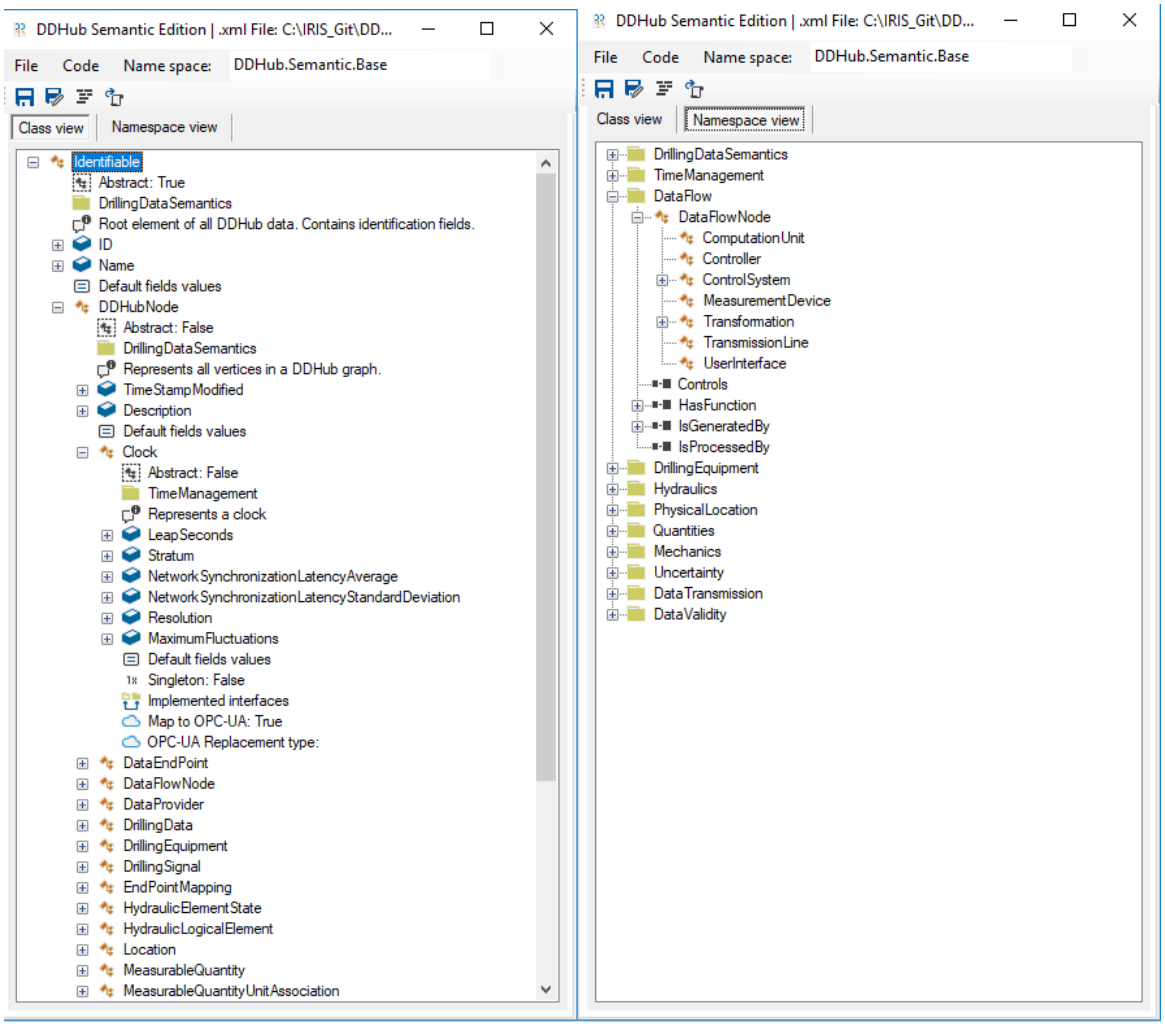

Figure 42: the tool for development and edition of the semantical model. The class view (left) displays the different types organized via their inheritance relations, while the

namespace view shows them organized by topic. Automatic generation of XML, C#, OWL2 and OPC-UA type libraries can be made as this level.

Model editor

It is necessary to be able to configure offline a DDHub graph: either a small component, corresponding to a single sensor, some base characteristic of a given installation (including units and quantities, hydraulic networks) or for demonstration purposes a full use-case. We developed a tool for that purpose, called DDHub model editor. The following tasks can be performed there:

- Import of existing graph
- Addition/removal of instances
- Addition/removal of relations
- Edition/modification of node's attributes
- Visualization of the DDHub graph
- Navigation in the graph by click combinations
- Export of the generated DDHub graph to different formats:
    - XML
    - C# generated code
    - OPC-UA node-set file
    - OWL2 data set

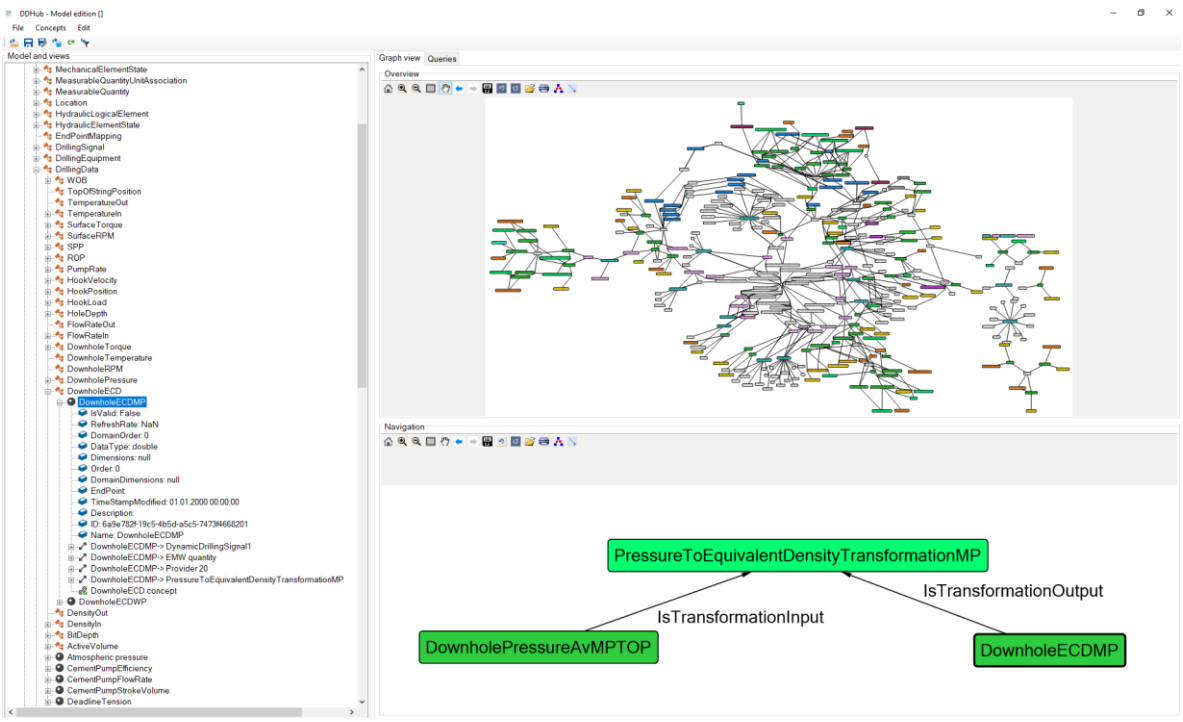

Figure 43: model edition tool. This application allows the off-line edition of a given model. Addition or removal of nodes and relations, edition of attributes can be performed with

the tool. Export of the DDHub graph to several formats (XML, C# code, OWL2, OPC-UA) is also implemented.

Service Company client

The main scenarios for drilling interoperability require the possibility to automatically connect to a server and add semantical information regarding one or several signals, as well as to populate the signals values. In other words, to fulfil the tasks of DDHub-S and DDHub-RT producer and provider.

We developed an application dedicated to those specific tasks, named temporarily the Service Company Client (Figure 44) because of the scenarios considered during our demonstrations. The functionalities of the application are:

- Load a sub-graph corresponding to a given drilling data (sensor value or computed one), including the relations to an existing DDHub graph,
- Automatically attach the sub-graph to the main central DDHub graph (if necessary, i.e. in the case where the considered data is not already present in the graph),
- Stream the values from the initial source to the DDHub server (using the OPC-UA implementation).

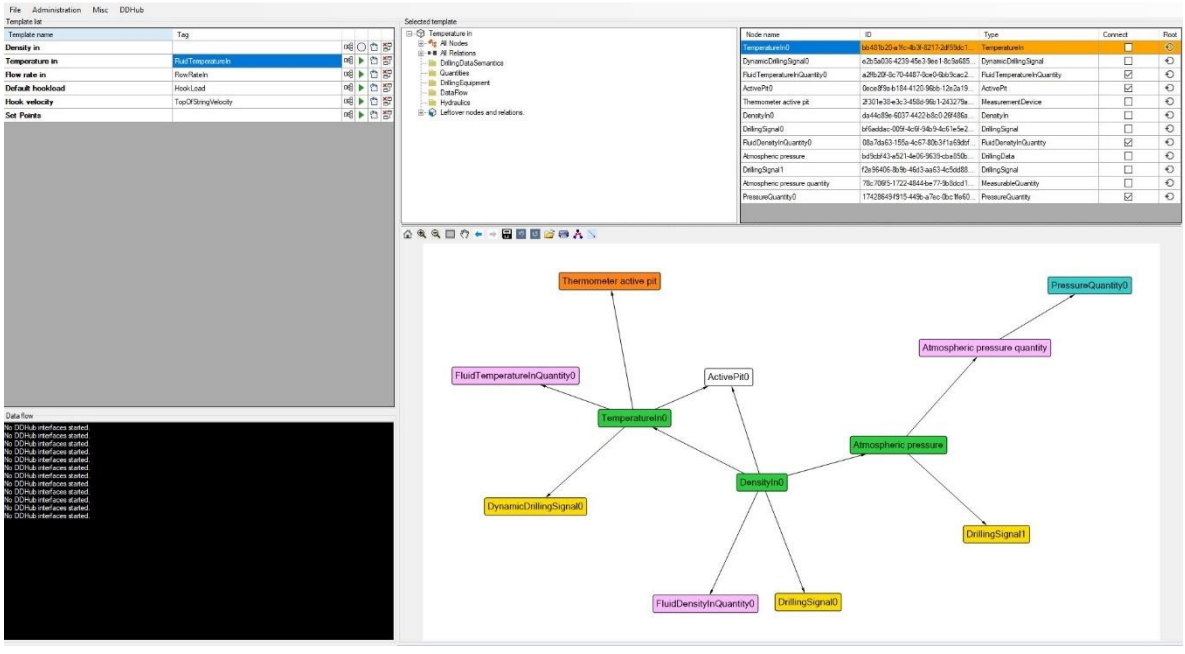

Figure 44: the Service Company Client, the application used to integrate signals (in particular sensor values) into a DDHub server.

The second task (attaching a sub-graph to the main one) is indeed a relatively complex one: its main objective is to recover the identifier of the main drilling data node. If the node is already present in the graph (because the application or sensor has already been integrated and needed to restart, for example), then one has to find it. If it is not present one has to

instantiate it, and record the identifier. We use for this the notion of template. A template is formed by the following (see Figure 45):

- A DDHub sub-graph,
- A root node of the sub-graph, corresponding to the main drilling data currently being integrated
- A list of nodes in the sub-graph whose presence is expected to be found in the main DDHub graph (for example quantity nodes).

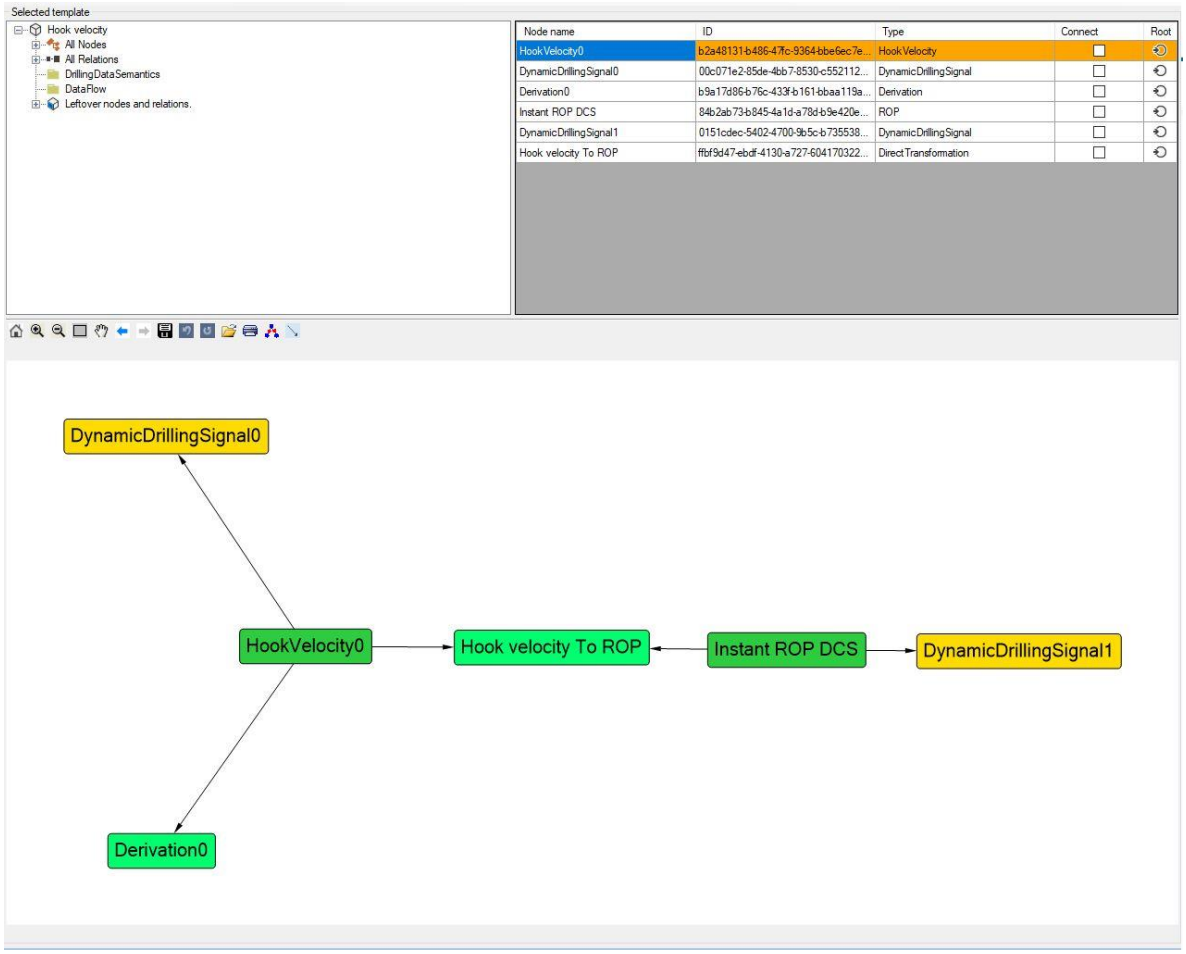

Figure 45: a template, with the sub-graph (displayed at bottom), the root node (highlighted in orange in the table at the top-right) and the nodes expected to be found in the central DDHub server (checked nodes in the table, none in the current example).

Then, at first a query is automatically generated from the sub-graph. If this sub-graph already exists in the main graph, the query will return the main drilling data node, and therefore the identifier can be recovered. If the sub-graph is not recognized, the system searches for all nodes expected to exist, using type-based queries. The results are stored, and the "missing" parts (nodes and relations) of the sub-graph are then added to the central one. During this latter adding process, the identifier of the main drilling data is generated and recorded. At this stage, independently of the initial presence or non-presence of the

considered signal, the identifier has been recovered, and streaming of values can be initiated.

In our implementation, automatic streaming of data is possible from the following sources:

- Wits
- UDP stream
- External OPC server
- openLab web interface (this source being of course only useful for demonstration purposes).

12. Demonstrations

Several demonstrations took place during the course of the project. They all had different purposes. The focus switched from dissemination objectives for the first demonstrations to presentation of the work done for the final ones. We briefly review the content of each of them.

October 2017

The very first demonstration was merely a comparison of the data management workload between the classical WitsML technology and the proposed semantical modelling. The advantages of automatic configuration of real-time signals were illustrated through various examples, as opposed to the need (in the case of WitsML) for manual monitoring of the real-time server and for manual configuration.

June 2018

In June 2018, the demonstration aimed at illustrating the potential of including semantical information to the classical set of real-time signals.

To this purpose, the demonstration used a scenario similar to the one sketched in chapter 5. There were a few noticeable differences though: more signals were involved (the classical rig signals, top-drive RPM, mud-pump rate, … were also covered), and set-down weights were the target of the detection algorithms (instead of over-pulls). In addition to detection, root-cause identification of the problem was also considered. Specifically, the system distinguished between set-down weights caused by ledges and alternatively cuttings accumulation.

Downhole ECD from wired pipe is available

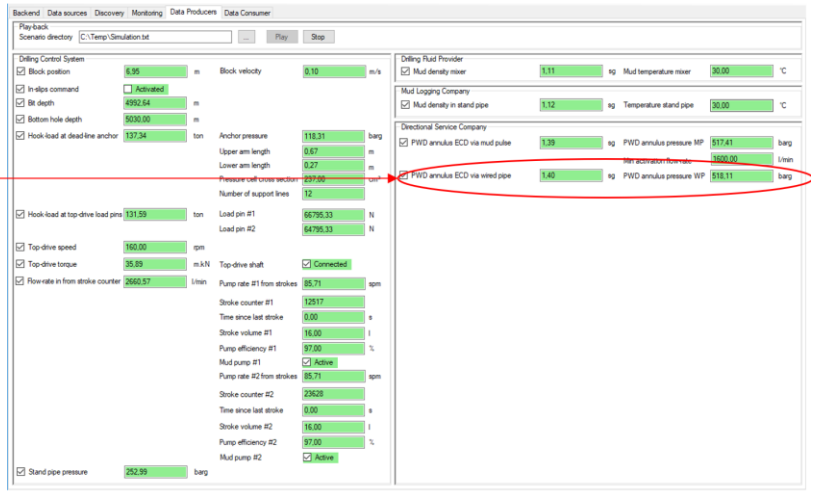

Figure 46: the event detection and root cause identification tool. The green boxes correspond to the different signals that are successively added to the DDHub server.

With downhole ECD transmitted with high speed telemetry, the system detects immediately that the cause of the abnormal hook-load is due to a pack-off

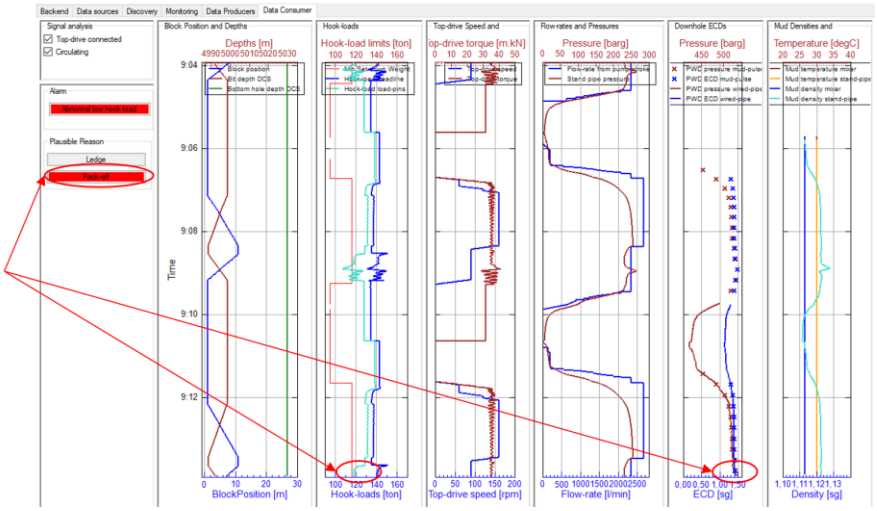

Figure 47: the simulation sequence used for the event detection and root cause identification scenario. In this example, the tool used all the information available to finally come to the (correct) conclusion that the set-down weight observed at the end of the sequence is due to a pack-off. The analysis used the fact that there is a simultaneous increase in downhole pressure.

December 2018

At that date, a first version of the semantical model was available, as well as first implementations of the various interfaces. It was then possible to start introducing the rest of the drilling community to the specifics of our approach. We organized a 2 day DDHub tutorial at the end of December 2018. Over 30 participants attended, with some of them travelling specifically for the occasion from the USA, Denmark, Germany or France.

The first day was devoted to general presentation of the work done and of the chosen approach, namely semantical modelling. A global review of the objectives was performed, deliveries of the project was done, and more technical issues were then addressed during the second day.

While we were expecting attendants familiar with our work and approach – we were assuming that the participants had attended the previous demonstrations—it turned out that many of them were completely new to our work. We had to considerably adapt our original schedule to this unforeseen situation, reflecting the fact that the DDHub project is difficult to present. The challenges related to proper interpretation of drilling signals are not always clearly identified by the industry, and the need for advanced solutions is not always accepted. However, during this 2-days session, fruitful discussions took place that helped refine the communication around the project.

June 2019

The June 2019 demonstration had as a main objective to reach a proof of concept for the ability of the DDHub to enable drilling interoperability. We therefore focused on automated interaction between several systems, which were:

- Sensor providers: several sets of sensory data were successively exposed to the DDHub system. The measurements themselves were extracted from the openLab simulator (https://openlab.app/). We considered 3 main signal groups, all managed by NORCE:
    - Rig signals, as provided by a rig contractor
    - Downhole signals, as typically provided by a service company
    - Additional rig signals (in our case more precise hook-load), as exceptionally provided by a third company.
- Drilling HMI: this corresponded to the signals generated from the drilling chair by the driller. A simple human/machine interface was developed by NORCE for the purposes of the demonstrations, that allowed to manually control the main machines: draw-works, top-drive and mud-pumps.
- Supervisory control: this part corresponds to a drilling control system. Its main function in this context is to generate set-points that are communicated to the machines themselves. For the demonstration, this meant that the set-points were streamed to the openLab simulator. This supervisory control was developed by NORCE and implemented the following logic: it collected all the set-point recommendations issued by the different actors. These included the set-points generated by the driller through the drilling HMI, and those generated by automated systems of Halliburton and Sekal. Manual selection (by an operator using a dedicated interface) was used to decide which of the available set-points would be finally sent to the machinery.

- DAS 1 (Halliburton): the system provided by Halliburton was a drilling advisory system that generated suggestions for drilling set-points, namely ROP, top-drive RPM and pump-rate.
- DAS 2 (Sekal): Sekal ran their DrillTronics system in a passive mode. In this context, this system provided real-time simulations of the main drilling variables (SPP, hook-load, surface torque, downhole pressures and forces) as well as recommendations for drilling parameters (here as well ROP, top-drive RPM, flow-rate).
- DAS 3 (Kongsberg): the last system was purely a monitoring application that automatically populated a WitsML server, and subsequently provided visualization interfaces to the ongoing drilling operation.

None of the various actors had any prior knowledge of the available signals and they all complied with the integration scenario described in section 8. In details, this means:

- Sensor providers: to each sensor value was attached a template, describing the measurement itself and the way it interacted with the remaining of the set-up. A dedicated tool was developed for the management of such templates (see Figure 48). The steps for adding a new sensor value were:
    - Identify if the sensor is already present in the DDHub server. This is not by automatically generating, for a given template, a query that allows to check if the pattern defined by the signal is already present or not.
    - If the signal is present, then the measurement provider knows where to subsequently write the signal's values.
    - If not, it then search (by using predefined queries) the elements of the graph it relates to and add the signal itself and the different necessary relations to existing elements.
    - Once the placeholder for the signal is in place, start streaming the sensor values to the DDHub server.
- Drilling HMI: this actor defines three main signals, namely set-point suggestions for draw-works, mud-pumps and top-drive. As for the sensor provider, it starts by searching for placeholders for those signals. Here as well, dedicated queries are used. In this case, the main characteristics are that the signals need to be provided by the "Driller" data provider, and that the signals need to be related to the main control systems by "IsSetPointRecommendation" relations. There are then two cases to identify the correct placeholder:
    - The signal is already present: then the placeholder is discovered by the query,
    - The signal is not present: then the signals are added as well as the necessary relations.
- Supervisory control: the same procedure is used to identify the placeholders for the set-points that are sent to the machinery. In addition, the system continuously monitors the DDHub server to keep an updated list of possible set-points recommendations. This is done by using the DDHub notification capabilities. Any

time a change is made to the DDHub server, the supervisory control identifies the set-point recommendations by using an adequate query. Those recommendations are then exposed to an operator via a dedicated interface.

- DAS 1: this system also starts by identifying the placeholders for the signals it will provide. It follows then the strategy of the Drilling HMI, since it also delivers set-point recommendations. It also uses dedicated queries to identify the different signals it needs to properly run, namely the classical rig signals, as provided by the first sensor provider.
- DAS 2: this system works as the DAS 1, except that it provides more signals. In addition to the set-point recommendations, it also provides simulated values for the main classical drilling signals (SPP, hook-load, surface torque, active volume…). Here as well, query-based placeholder identification is used. Just as the DAS 1, it uses signals provided by the different systems to run properly, but implements a more advanced logic: when several alternatives are available, it automatically chooses which input signal suits best its purpose. It therefore constantly monitors the DDHub server via the notification interfaces, and possibly modifies its sources of information when better measurements become available (such as when a top-drive based hook-load is added to the system, making the dead-line signal superfluous).
- DAS 3: this system has a built-in selection mechanism, that automatically decides which signal to display to the end-user depending on the signal's provider. It therefore continuously monitors the DDHub server, gathers the different relevant measurements, and sorts them according to this prioritization. As a result, the final display fully accounts for the semantics that the different signals carry.

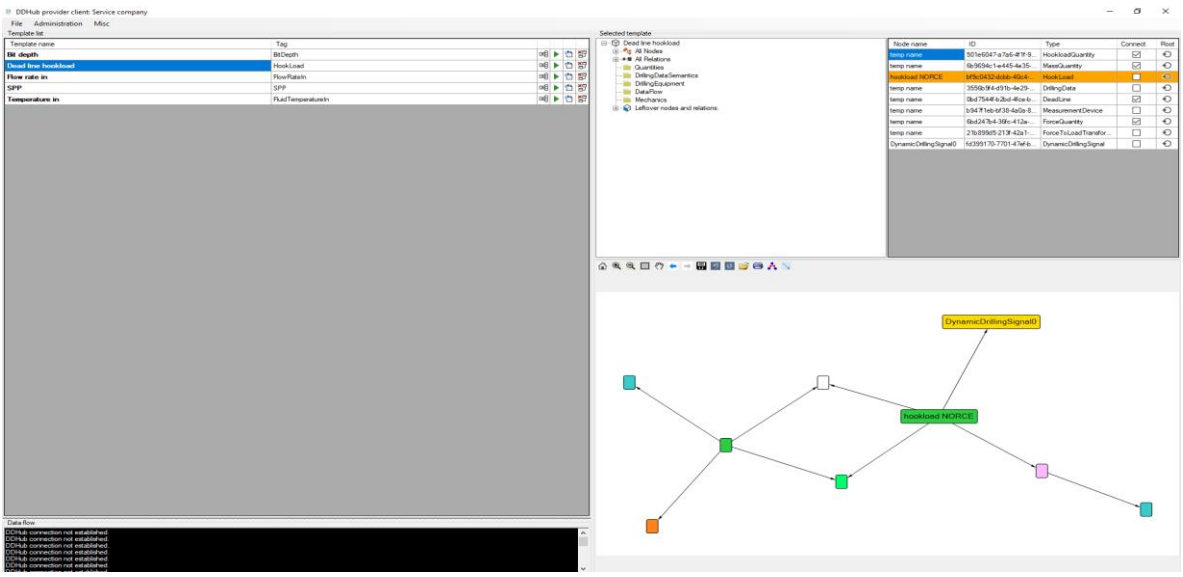

Figure 48: example of a sensor provider graphical interface.

The DDHub server was populated incrementally: the sensors and HMI values were added one by one. As a result, the other systems reacted dynamically: DAS 1 and DAS 2 started

automatically when a sufficient amount of information was present in the system, and adapted their own internal acquisition system when better suited signals were added. The Supervisory control application automatically detected when the DAS 1 and DAS 2 started to provide set-point recommendations, and the DAS 3 system modified the WitsML display when signals with higher priority entered the DDHub.

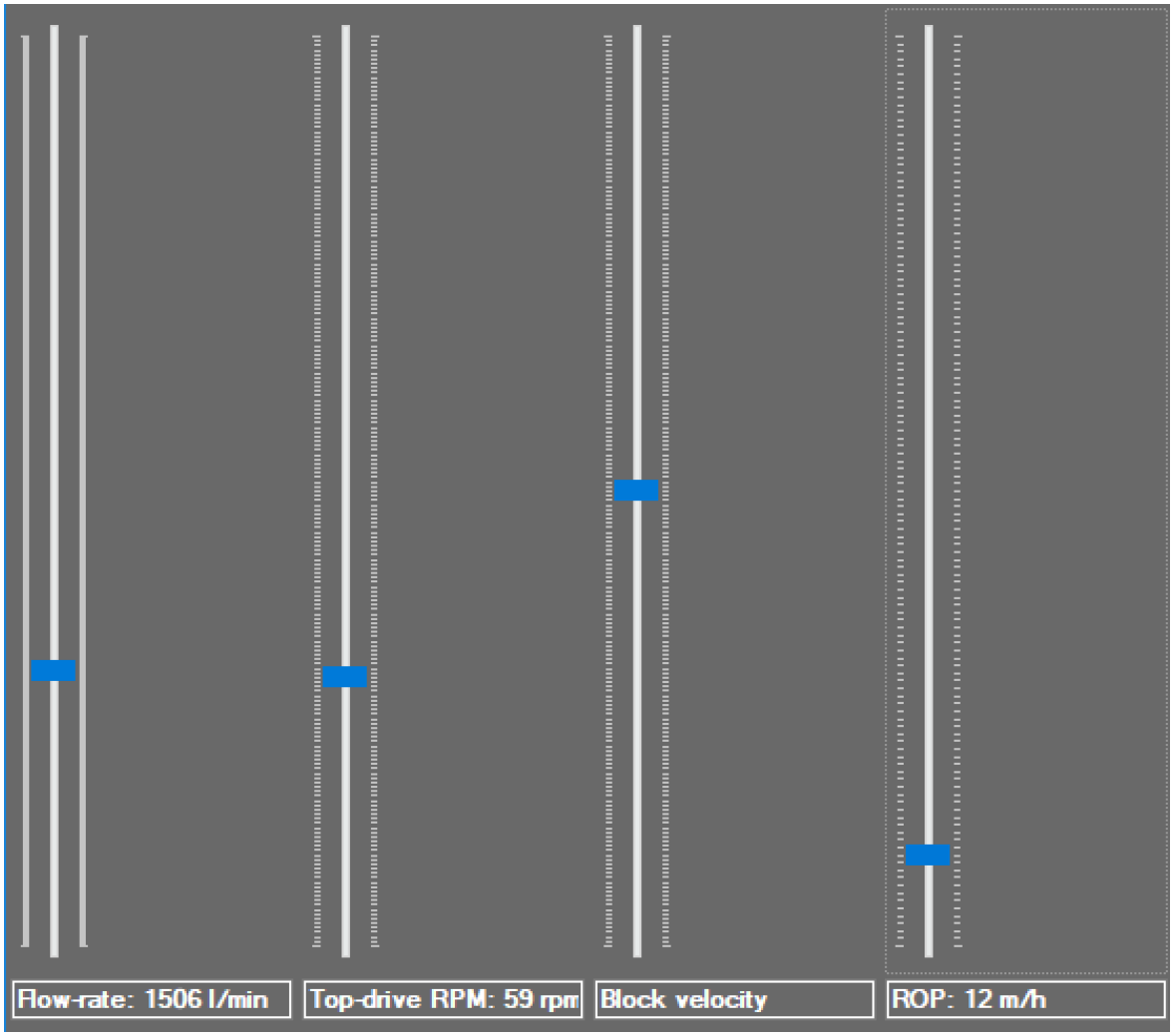

Figure 49: the drilling HMI interface

The set-up of the demonstration was quite simplified compared to a real drilling installation, but nevertheless this demonstration proved that it is possible for different components to work together as intended, without extensive manual configuration work. The reactivity required for interoperability was also demonstrated.

December 2019

The latest demonstration of the project focused on the aggregation capabilities of our system. The parties involved were the same as for the previous demonstration. The main differences were in the way the various signals were transferred and integrated into the DDHub framework.

We considered several sources of signals: WITS stream, direct OPC communication, UDP link with the drilling control system or DDHub communication interfaces (see Figure 50). This disparity in sources lead to a disparity in the procedures to integrate the measurement's semantical description. Several integration procedures and interfaces were therefore considered:

- Downhole signals: the signals themselves were added to the OPC-UA server using native OPC-UA interfacing. The DDHub-RT Producer function did therefore not involve any DDHub technology. The integration into the DDHub framework was performed at a later stage, using a dedicated tool (developed by NORCE). Taking advantage of the specifics of the OPC-UA implementation of the DDHub model, (see chapter 10), DDHub-S Producer could easily refer to (and fully integrate) the downhole signals into the DDHub server.
- Rig signals: two versions of the rig signals were exposed. A classical Wits0 stream was replicated into the DDHub server, while a high-speed stream was also made available. Both sets of signals corresponded to the same set of sensors, but the acquisition rates were different. In both cases, an operator used the dedicated graphical tools to perform simultaneously the DDHub-RT and DDHub-S producer tasks.
- Set-points: the set-points used by the controllers (and generated by the drilling control system) were exposed to the DDHub server via direct access to the drilling control system, through an UDP connection. Here as well, the semantics of the signals was included by an operator at the same time as adding the signals in the server.
- Set-point recommendations and simulation values: these were the signals generated by the different DAS involved in the demonstration. The DDHub-RT and DDHub-S functions were fulfilled automatically (*i.e.* without human intervention), but via two different communication interfaces: one DAS used the C# SDK, while the other one relied on the REST web API.

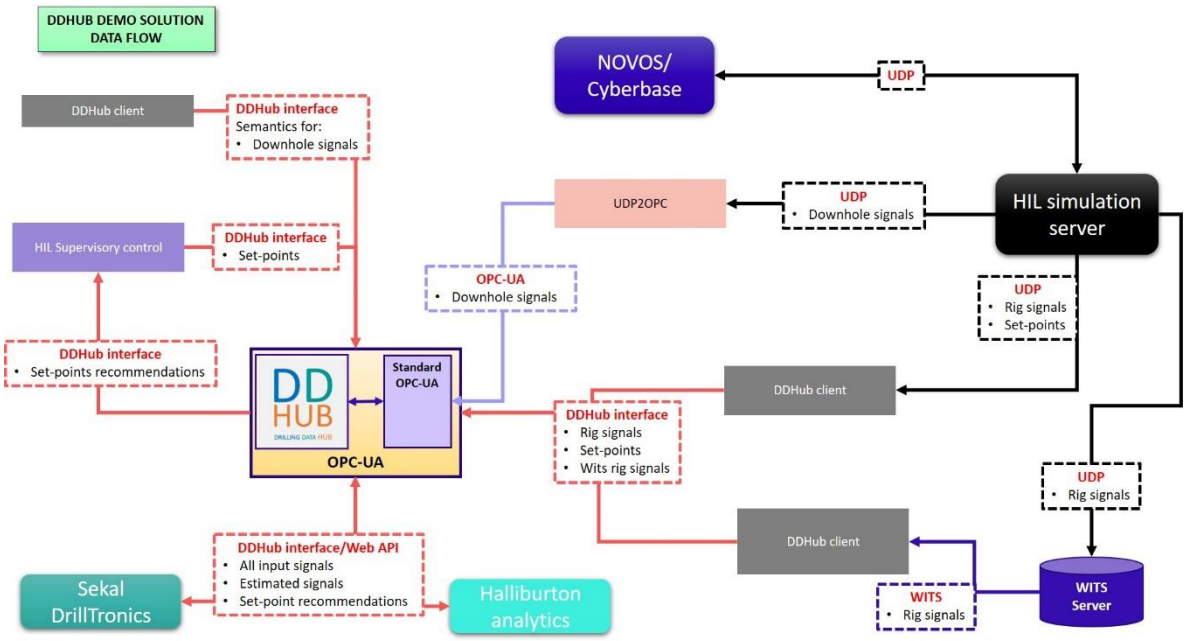

Figure 50: the topology of the demonstration. All signals converged ultimately to the DDHub server (and its OPC-UA implementation) but took different paths. Access to the server was also done in different ways: native OPC, C# SDK, Web API.

In spite of the disparity of configurations, all the involved systems managed to fulfill their respective tasks.

Apart from the various integration possibilities, the synchronization capabilities described in chapters 9 and 10 were illustrated with automated interpretation of (delayed) downhole data.

13. Distribution

The work done in this project is open source, and available to any person interested. In order to introduce, explain, and distribute our project a web-site has been created, available at https://ddhub.no/.

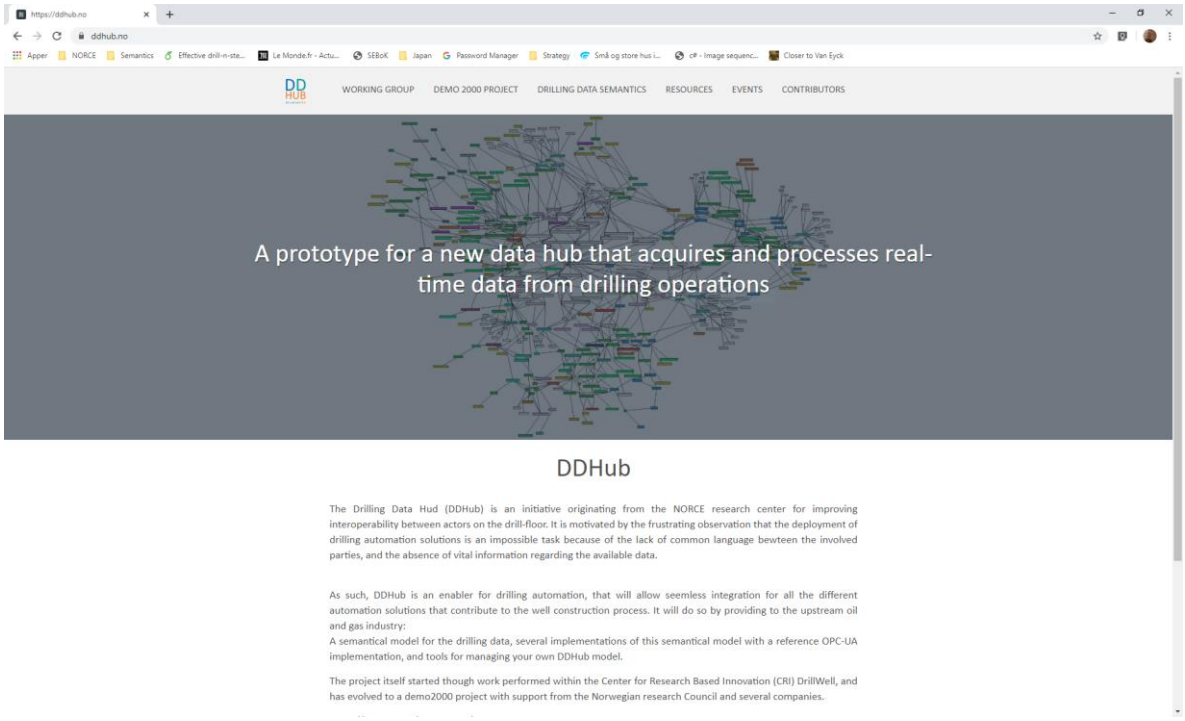

Figure 51: the DDHub web-site

The web-site contains basic explanations about the chosen approach, but serves for the moment as a main entry point to other dissemination systems:

- A GitHub repository that contains all the code written during the project
- Basic documentation
- A NuGet package containing the main library to be used by the C# SDK
- Links to the main publications attached to the project, namely (Cayeux, Daireaux, Saadallah, & Alyaev, 2019) and (Suter, Alyaev, & Daireaux, 2017).

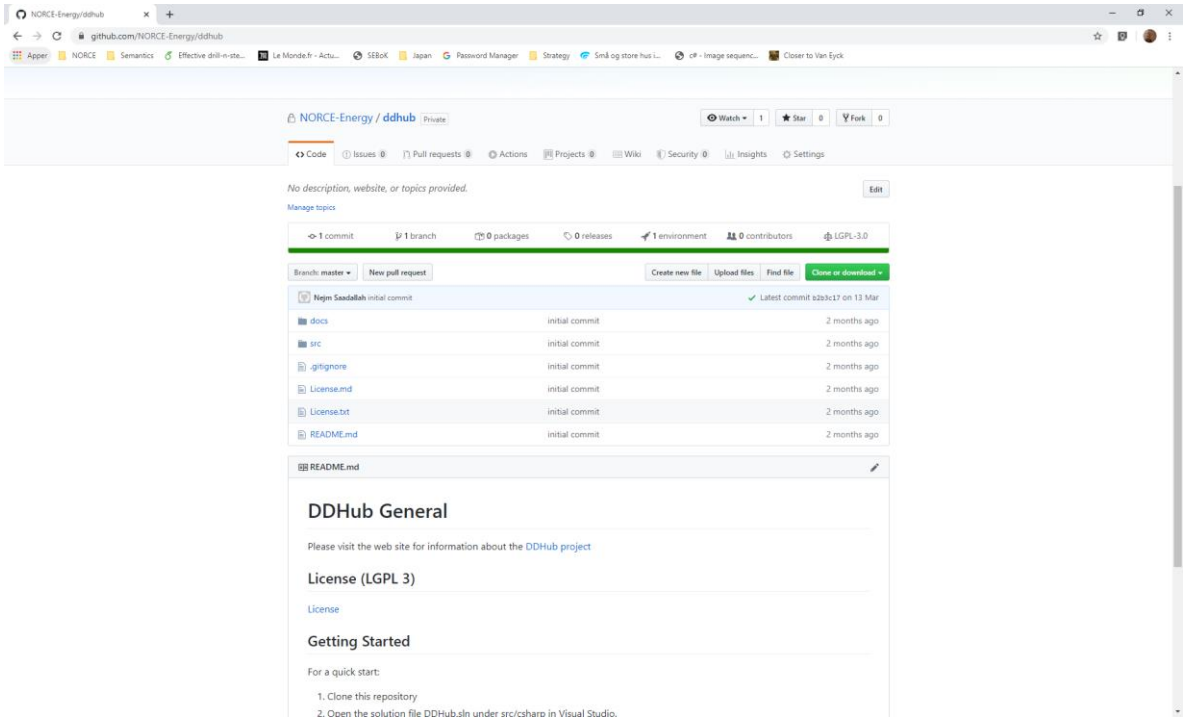

Figure 52: the DDHub GitHub repository

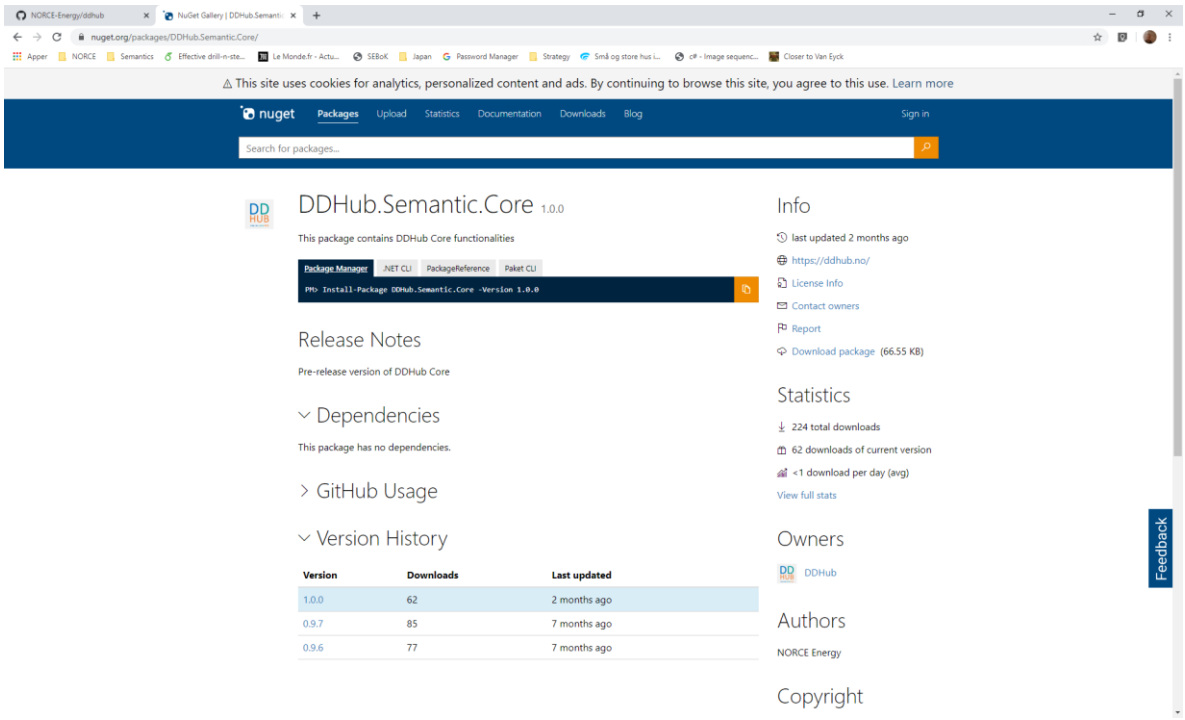

Figure 53: The DDHub NuGet package

14. Conclusion

The main objective of the project was to perform a proof of concept of data interoperability within drilling operations. This required several developments, from a base formalism for the representation of the information about drilling data, specialization of this formalism to the drilling domain, implementation of software systems, and tool to manipulate, exchange, and discover this information, all within the real-time context that characterizes drilling automation systems.

The different demonstrations involved several actors from the drilling industry, and proved that the chosen approach indeed enabled interoperability: in all cases, the different systems were able to perform their tasks, sometimes in close collaboration with the other actor, without any prior knowledge of the available drilling data. All the integration processes between the many components was fully automated, corresponding to interoperability.

The demonstrations took place in a virtual environment, and not on a real drilling rig. A natural further step would be to consider piloting the various prototypes in real drilling operations.

Further suggestions to continued work:

- The semantical definitions developed in the project all originate from NORCE. For such an initiative to be adopted by the industry, it is necessary to involve other participants who can refine, extend and modify the base semantical model. Such work can be hosted by an industry-wide organization. The SPE D-WIS committee is a natural candidate for such an activity.
- The prototypes implemented can be developed into industrial products. In order to facilitate the use of the system, robust implementations are necessary, and this requires assistance from professional software companies.
- Demonstrations: acceptance by the industry relies also on demonstrations of the benefits that interoperability can provide. Such demonstrations should involve more companies, and could target different aspects of the drilling operations, such as:
    - Drilling analytics systems,
    - Geo-steering
    - Active machine control
    - Interfacing to drilling control system interfaces
    - Integration of drilling procedures in the automation systems

Initiatives for continuing these activities are ongoing.

15. References

Annaiyappa, P. (2013). IT Infrastructure Architectures to Support Drilling Automation. *SPE Digital Energy Conference and Exhibition.* The Woodlands, Texas, USA.

Bauer, M. (2019). *Semantic IoT Solutions - A Developer Perspective.*

Bauer, M. (2019). *Towards Semantic Interoperability Standards.*

Baumgartner, T., Ashok, P., & van Oort, E. (2019). Automated Preprocessing Techniques for High Frequency Downhole Sensor Data. *SPE/IADC International Drilling Conference and Exhibition.* The Hague, The Netherlands.

Baumgartner, T., Ashok, P., & van Oort, E. (2019). Automated Preprocessing Techniques for High Frequency Downhole Sensor Data. *SPE/IADC Drilling International Conference and Exhibition.* The Hague, NL.

Baumgartner, T., Zhou, Y., & van Oort, E. (2016). Efficiency Transferring and Sharing Drilling Data from Downhole Sensors. *IADC/SPE Drilling Conference and Exhibition.* Fort Worth, TX.

Baumgartner, T., Zhou, Y., & van Oort, E. (2016). Efficiently Transferring and Sharing Drilling Data from Downhole Sensors. *IADC/SPE Drilling Conference and Exhibition.* Fort Worth, Texas, USA.

Behounek, M., Hofer, E., Thetford, T., White, M., Yang, L., & Taccolini, M. (2017). Taking a Different Approach to Drilling Data Aggregation to Improve Drilling Performance. *SPE/IADC Drilling Conference and Exhibition.* The Hague, NL.

Behounek, M., Thetford, T., Yang, L., Hofer, E., White, M., Ashok, P., . . . Ramos, D. (2017). Human Factors Engineering in the Design and Deployment of a Novel Data Aggregation and Distribution System for Drilling Operations. *SPE/IADC Drilling Conference and Exhibition.* The Hague, NL.

Borges, J. L. (1942). Celestial Emporium of Benevolent Knowledge. I J. L. Borges, *John Wilkins' Analytical Language.* Penguin Books.

Cayeux, E., Daireaux, B., & Dvergsnes, E. (2011, December). Automation of Drawworks and Topdrive Management To Minimize Swab/Surge and Poor-Downhole-Condition Effect. *SPE Drilling & Completion, 26*(04).

Cayeux, E., Daireaux, B., & Dvergsnes, E. (2011, March). Automation of Mud-Pump Management: Application to Drilling Operations in the North Sea. *SPE Drilling & Completion, 26*(01).

Cayeux, E., Daireaux, B., Saadallah, N., & Alyaev, S. (2019). Toward Seamless Interoperability Between Real-Time Drilling Management and Control Applications. *SPE/IADC International Drilling Conference and Exhibition.* The Hague, NL.

Cayeux, E., Mihai, R., Carlsen, L., & Stokka, S. (2020). An Approach to Autonomous Drilling. *IADC/SPE International Drilling Conference and Exhibition.* Galveston, Texas, USA.

Cayeux, E., Skadsem, H., & Kluge, R. (2015). Accuracy and Correction of Hook Load Measurements During Drilling Operations. *SPE/IADC Drilling Conference and Exhibition.* London, UK.

Chmela, B., Abrahmsen, E., & Haugen, J. (2014). Prevention of Drilling Problems Using Real-Time Symptom Detection and Physical Models. *Offshore Technology Conference-Asia.* Kuala Lumpur, Malaysia.

Daireaux, B., Dvergsnes, E., Cayeux, E., Bergerud, R., & Kjøsnes, I. (2019). Automatic Control of Mud Pumps, Draw-Works and Top-Drive on a Floater. *SPE/IADC International Drilling Conference and Exhibition.* The Hague, The Netherlands.

DSA-R. (2019). *Drilling system Automatin Roadmap Report.*

Dunlop, J., Isangulov, R., Aldred, W. D., Arimsmendi Sanchez, H., Sanchez Flores, J. L., Alarcon Herdoiza, J., . . . Luppens, J. C. (2011). Increased Rate of Penetration Through Automation. *SPE/IADC Drilling Conference and Exhibition.* Amsterdam, NL.

Dwars, S. (2015). Recent Advances in Soft Torque Rotary Systems. *SPE/IADC Drilling Conference and Exhibition.* London, UK.

Dwars, S., Lien, M., Øydna, S., & Baumgartner, T. (2019). Curing stick-slip: Eureka. *SPE/IADC International Drilling Conference and Exhibition.* The Hague, The Netherlands.

Energistics. (u.d.). *WitsML*. Hentet fra https://www.energistics.org/portfolio/witsml-data-standards/

Gasser, U., & Palfrey, J. (2008). *Breaking down digital barriers: When and how ICT interoperability drives innovation.* Berkman Center Research Publication .

Goncalves, K., Ashok, P., Cavanaugh, M., Macpherson, J., Behounek, M., Thetford, T., & Nelson, B. (2017). A Meta-Data Framework for Transparency in Rate of Penetration Calculations. *SPE Annual Technical Conference and Exhibition .* San Antonio, TX.

Isbell, M. (2017, November). Unsynced time measurements can lead to data aggregation challenges. *Drilling Contractor*. Hentet fra https://www.drillingcontractor.org/unsynced-time-measurements-can-lead-to-data-aggregation-challenges-44767

Janowicz, K., Haller, A., Cox, S. J., Le Phuoc, D., & Lefrancois, M. (2019). SOSA: A lightweight ontology for sensors, observations, samples, and. *Web Semantics: Science, Services and Agents on the World Wide Web*(56), 1-10.

Kharlamov, E., Hovland, D., Skjæveland, M., Bilidas, D., Jimenez-Ruiz, E., Xiao, G., . . . Waaler, A. (2017). Ontology Based Data Access in Statoil. *Web Semantics: Science, Services and Agents on the World Wide Web, 44*, 3-36.

Kharlamov, E., Mailis, T., Mehdi, G., Neuenstadt, C., Ôzcep, Ö., Roshchin, M., . . . Waaler, A. (2017). Semantic access to streaming and static data at Siemens. *Web Semantics: Science, Services and Agents on the World Wide Web, 44*.

Kyllingstad, Å., & Nessjøen, P. (2009). A New Stick-Slip Prevention System. *SPE/IADC Drilling Conference and Exhibition.* Amsterdam, The Netherlands.

Luo, X., Eliasson, P., Alyaev, S., Romdhane, A., Suter, E., Querendez, E., & Vefring, E. (2015). An Ensemble-Based Framework for Proactive Geosteering. *SPWLA 56th Annual Logging Symposium.* Long Beach, CA, USA.

Macpherson, J. D., de Wardt, J. P., Florence, F., Chapman, C. D., Zamora, M., Laing, M. L., & Iversen, F. P. (2013). Drilling-Systems Automation: Current State, Initiatives, and Potential Impact. *SPE Drilling & Completion*, 296-308.

Macpherson, J., de Wardt, J., Laing, M., & Zenero, N. (2016). Data Ownership for Drilling Automation - Managing the Impact. *IADC/SPE Drilling Conference and Exhibition.* Fort Worth, Texas, USA.

Macpherson, J., Pastusek, P., Behounek, M., & Harmer, R. (2015). A Framework for Transparency in Drilling Mechanics and Dynamics Measurements. *SPE Annual Technical Conference and Exhibition.* Houston, TX.

Maus, S., Gee, T., Mitkus, A., McCarthy, K., Charney, E., Ferro, A., . . . Mottahedeh, R. (2020). Automated Geosteering with Fault Detection and Multi-Solution Tracking. *IADC/SPE International Drilling Conference and Exhibition.* Galveston, Texas, USA.

Mayani, M., Rommetveit, R., Oedegaard, S., & Svendsen, M. (2018). Drilling Automated Realtime Monitoring Using Digital Twin. *Abu Dhabi International Petroleum Exhibition and Conference.* Abu Dhabi, UAE.

Murdock, P. (2016). *Semantic Interoperability for the Web of Things.*

Nafikov, R., & Glomstad, S. (2013). Automatic Mud Mixing. *SPE/IADC Drilling Conference and Exhibitin.* Amsterdam, NL.

Payette, G. S., Spivey, B. J., Wang, L., Bailey, J. R., Sanderson, D., Kong, R., . . . Eddy, A. (2019). A Real-Time Well-Site Based Surveillance and Optimization Platform for Drilling: Technology, Basic Workflows and Field Results. *SPE/IADC Drilling Conference and Exhibition.* The Hague, NL.

Sadlier, A., & Laing, M. (2011). Interoperability: An Enabler for Drilling Automation an a Driver for Innovation. *SPE/IADC Drilling Conference and Exhibition .* Amsterdam.

Sadlier, A., Laing, M., & Shields, J. (2012). Data Aggregation and Drilling Automation: Connecting the Interoperability Bridge between Acquisition, Monitoring, Evaluation and Control. *IADC/SPE Drilling Conference and Exhibition.* San Diego.

Shields, J., & Brackel, H. (2014). Improving Wellsite Systems Integration. *SPE/ICoTA Coiled Tubing & Well Intervention Conference and Exhibition.* The Woodlands, TX.

Simmons, F. R. (1963). Synthetic language behavior. *Data Processing Management, 5*(12), 11-18.

Sletcha, B., Vivas, C., Saleh, F., Ghalambor, A., & Salehi, S. (2020). Digital Oilfield: Review of Real-time Data-flow Architecture for Upstream Oil and Gas Rigs. *SPE International Conference and Exhibition on Formation Damage Control.* Lafayette, Louisiana, USA.

Spivey, B. J., Payette, G. S., Wang, L., Bailey, J. R., Sanderson, D., Lai, S. W., . . . Eddy, A. (2017). Challenges and Lessons from Implementing a Real-Time Drilling Advisory System. *SPE Annual Technical Conference and Exhibition.* San Antonio, TX.

Suter, E., Alyaev, S., & Daireaux, B. (2017). RT-Hub - Next Generation Real-time Data Aggregation While Drilling. *First EAGE Workshop on Pore Pressure Workshop.*

Taugbøl, K., Brevik, J., & Rudshaug, B. (2019). Automatic Drilling Fluid Measurements. *SPE Russian Petroleum Technology Conference.* Moscow, RU.

WITS. (u.d.). *Wellsite Information Transfer Specification*. Hentet fra http://www.petrospec-technologies.com/resource/wits_doc.htm

16. Index