NTNU

Norwegian University of
Science and Technology

Analysis of Integration Architecture in Integrated Environmental Monitoring System

Development of Front-End Prototype for
Decision Support for Drilling in Areas with
Sensitive Fauna

Therese Torgersen

Master of Science in Engineering and ICT
Submission date: February 2017
Supervisor: Vidar Hepsø, IGP
Co-supervisor: Tor Nordam, SINTEF

Norwegian University of Science and Technology
Department of Geoscience and Petroleum

To
Vilde Mari Torgeresen,
Marianne Torgersen,
and
Torgeir Torgersen

Abstract

This master's thesis is a continuation of the specialization project, which focused on the underlying need for better surveillance of sensitive fauna, during drilling operation, and an industry developed software solution designed to meet this need - ELMO. ELMO is an integrated environmental monitoring software solution that consist of several preexisting systems. By adopting an integration architecture ELMO is able to collect data from several different sources and present the user with valuable information about the risk imposed on corals during drilling operations.

This master's thesis is divided into a theoretical part and a practical part. The objective of the theoretical part is to develop a framework for analysing integration architectures in systems-of-systems, and then apply the framework to analyse ELMO's integration architecture. The purpose of the practical part, is to acquire relevant skills and develop a prototype for a user interface that is intended to enhance ELMO's value as a decision support tool during drilling operations.

Preface

This report is the result of the course TPG4935 Integrated Operation in the Petroleum Industry, Master's Thesis. The report was written in the period between September 2016 and February 2017 as a part of a master program in Engineering and ICT at the Norwegian University of Science and Technology.

The thesis consist of a framework for analysing integration architectures in systems-of-systems, an analysis of ELMO's integration architecture and a chapter describing a front-end prototype.

Acknowledgments

I would like to thank my responsible supervisor Vidar Heps for introducing me to Integrated Environmental Monitoring, and in particular to the ELMO-solution.

I would also like to thank my supervisor Tor Nordam from SINTEF Materials and Chemistry for his support, guidance and feedback during the project

Finally, thanks to Torgeir Torgersen for being a source of inspiration.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem description

To cope with an increasing interest to commence drilling in sensitive areas, the industry sees the need to introduce Integrated Environmental Monitoring. Integrated Environmental Monitoring is careful monitoring of drilling operations and the surrounding environment to prevent damages to sensitive fauna such as corals.

Environmental Monitoring and Modeling (ELMO) is an Integrated Environmental Monitoring software solution developed by Sintef and Statoil. ELMO comprises of three preexisting software systems, Sinmod, DREAM and C/XRA, in addition to a controller application, server solution and a user interface. Integration of the individual systems into a system-of-systems enables ELMO to use real-time data to indicate the risk profiles of corals during drilling operations.

To be able to integrate the different systems into a system-of-systems an integration architecture is required. An integration architecture enables efficient communication of data from several sources, cross-systems transactions and automation of processes.

This master's thesis theoretical task is to develop a framework for analysing integration architectures in systems-of-systems, and use the same framework to analyse the integration architecture used in ELMO.

The objective of the practical task is acquire knowledge and competence related to front-end development. This will then be used to implement a front-end prototype that displays the progress of a specified drilling operations as well as the accumulation of drilling discharges on a point-of-interest.

1.2 Motivation

Focusing on integration architecture analysis in the theoretical part, was a natural next step considering the analysis that were conducted in the specialization projects. It is also an opportunity to expand my knowledge and understanding of relationships between system

development, system architecture and system integration.

For the practical part of the master's thesis several solution were considered before deciding on developing a front-end prototype. The front-end prototype presented an opportunity to learn something completely new that matched my level of experience. Therefore, learning the mechanisms and contexts of front-end development is an equally important goal as creating the desired functionality.

1.3 Structure of thesis

The first chapter in this thesis is a summary of the findings of the literature review of the specialization project. It explains the importance of integrated environmental monitoring, and presents a concrete example of a solution - ELMO - developed by the industry to handle the challenges related to drilling in sensitive areas.

Chapter two contains a framework for analysing the integration architecture of systems-of-systems. The framework has been developed based on literature from several sources and aims to be a tool for analysing integration architectures in systems-of-systems. The ELMO solution is presented in chapter four, and is meant to provide the reader with necessary information about ELMO before ELMO's integration architecture is analysed in chapter 5.

Chapter six examines the practical task, by comparing the target environment (the desired result), with the prototype environment (the actual result). Chapter seven contains the conclusion for overall project and highlights the main points in the master's thesis.

Chapter 2

Background

This chapter is meant to provide the reader with a basic introduction to certain aspects of drilling operations, how these aspects may affect the environment surrounding the wellsite and the software solution made by the industry to cope with these challenges. The information presented in this chapter is a summary based on the specialization project, for a more comprehensive introduction the reader is referred to Torgersen (2016)

2.1 Drilling in sensitive areas

In 2016 206 production and exploration wellbores were spudded on the Norwegian Continental Shelf (Norwegian Petroleum Depeartmen 2016). A by-product of these drilling operatings are drilling discharges. Drilling discharges consists of drill cuttings (crushed material from the wellbore), drilling mud and chemicals. Drilling mud is a specially designed fluid that lubricates the drill bit, transport the drill cuttings and balances the formation pressure in the wellbore (Petrowiki.org 2015*a*). There are three types of drilling mud; water-based mud, oil-based and synthetic-based mud. The type of drilling mud and depth of the section in the wellbore determines how the drilling discharges are disposed (Brechan et al. 2016).

Usually the drill cuttings from the top hole are discharged at the sea floor in the vicinity of the well. As long as water-based mud is used, and less than one percent is contaminated by oil, the drill cuttings can be discharged at sea. If oil-based mud or synthetic based mud are used, and/or the cuttings are from deeper sections, the cuttings are separated from the drilling mud so that the drilling mud can be recycled and re-injected into the well, while the drill cuttings are discharged (Brechan et al. 2016).

When drill cuttings are discharged to sea this can be done from the rig, both above and below the sea surface, directly at the sea floor or by using a Cutting Transport System. A Cutting Transport System can be applied when there are sensitive fauna in the area of the wellbore, to protect the fauna from the drill cuttings.

On the Norwegian Continental Shelf there are several areas with sensitive fauna, such as cold water corals and sponges. Coral reefs are habitats to a rich set of species, and are

associated with higher biodiversity than the surrounding areas. One of the main threats towards coral reefs and corals associated with offshore petroleum activity is smothering as a result of accumulation of drilling discharges (Larsson & Purser 2011), (Foss et al. 2002).

Drilling operations follow a drilling plan describing every aspect of the operations. Drilling plans are developed by professionals before the operations and contains information about the shape, orientation, depth, completion and evaluations of the drilling operation (Petrowiki.org 2015*b*). The drilling plan also predicts expected volumes of mud and cuttings and the duration for each section (DNV GL 2014). Each section in a well requires three steps; drilling, casing and cementing (Brechan et al. 2016). As a result discharges are intermittent.

2.2 Environmental Monitoring and Modeling, (ELMO)

When drilling in sensitive areas, it is a requirement from the Norwegian authorities that the operator shall monitor if and how the drilling operation affects surrounding areas (Rye et al. 2013). While off-line annual and triennial, monitoring is the standard for environmental monitoring now, the petroleum industry is entering areas with increasingly sensitive environments. This will require more flexible and real-time sensor based monitoring which can be tied closely to daily operations, also known as Integrated Environmental Monitoring (Hepsø et al. 2012).

In 2012 SINTEF Materials and Chemistry, SINTEF Fisheries and Aquaculture, and Statoil developed the ELMO prototype, a research project within the DEMO2000 frame of the Norwegian Research Council. ELMO is a software solution made to support the decision process by presenting real-time environmental data during drilling operations in sensitive areas (Sintef 2017*b*).

ELMO consists of a hydrodynamic model, a sedimentation model, and a risk assessment tool. To connect all these software models a controller application has been developed with an associated front-end application and THREDDS server. To ensure seamless interaction and data flow between the data sources, models and front-end application a set of open standards are adopted (Brönner et al. 2016).

Chapter 3

Framework for Analysing Integration Architecture in a System-of-Systems

A system-of-systems, is a system where several individual and independent systems are integrated to achieve cooperation and interoperation (Kazman et al. 2013). The purpose of integrating the systems is to meet new business needs that will increase the competitive advantage and enhance the operational efficiency by giving better insight into key business data and improve business support (Ahsan & Nurmilaakso 2015).

The most important feature in an integration process is the integration architecture. ”An integration architecture consists of principles, processes and technical solutions for managing the distribution and heterogeneity of application landscape and the underlying technologies” (Murer et al. 2011)

This chapter presents a comprehensive framework for analysing the integration architecture in a system-of-systems. During the literature review multiple sources covering sections of the overall integration architecture were studied. The framework presented combines theory from several sources, with the main contributor being *Understanding Patterns for System-of-Systems Integration* (Kazman et al. 2013), and *Integration Patterns* (Microsoft Corporation 2004), and aims to provide a foundation that can be used to analyse the overall integration architecture in a system-of-systems.

The framework addresses five main topics: System design, Scope and development context, Sharing of data, Control regime and System connection. Together these five topics explain why the design of individual systems determines how they are integrated into a system-of-system and presents several methods to analyse the underlying integration architecture.Scope and development context identifies the initial state of a system-of-systems and its component and the associated limitations and opportunities.

Sharing of data addresses how data is shared among systems, the need for an information model and how to implement a solution that can effectively provide unified access

to data that is scattered across multiple distributed repositories. Control regime describes the functionality needed to control operations and transactions than span multiple systems. System connections presents the different ways systems can connect to each other.

3.1 System Design

A system's design determines if a system can be integrated in to a system-of-systems and how the system are to be connected to the other systems. A system may be little receptive to integration, as in the case of monolithic systems. Monolithic systems are tightly coupled with one system instance in charge of everything; user interfaces, accessing and validating data, and core functionality (Stephens 2015). The only way to integrate such systems are either by comprehensive reengineering or by accessing the user interface through presentation integration.

Other systems presents several possible connection points, such as systems adopting a layered system design. A layered system design separates responsibilities into layers and strive to limit the dependencies between each layer. The number of layers vary, and depends, among other things, on the complexity of the system. For the reaming part of this thesis, layered system design refers to the three-layered design. Each of the three following layers, as defined by Fowler (2002), provides a connection point for other systems

- **Presentation Logic Layer**: Is in charge of communicating with the user by displaying information and interpret commands from the user and bring them to the application logic. This can be achieved using everything from simple command line interfaces to rich clients or web interfaces.
- **Application Logic Layer**: Contains the applications functionality and utility components that are used by other application components. The application layer is also in charge of system security and validating user input and data received from other applications.
- **Data Source Layer**: Is in charge of communication with databases moving information to and from the databases and handles transaction management (Sommerville 2011).

Figure 3.1 presents graphical representations of a monolithic and layered system design and the differences between them. Although layered system design is one of the most commonly used designs, there are many systems, ELMO included, that adopts other kinds of system design. But using the three above mentioned logical layers presents a common way to analyse system connections regardless of system design.

3.2 Scope and Development Context

The scope and development context identifies the initial state of the different pieces of the puzzle that becomes a completely integrated systems-of-system. Most organizations have systems developed over longer periods of time, and as a result most systems-of-systems

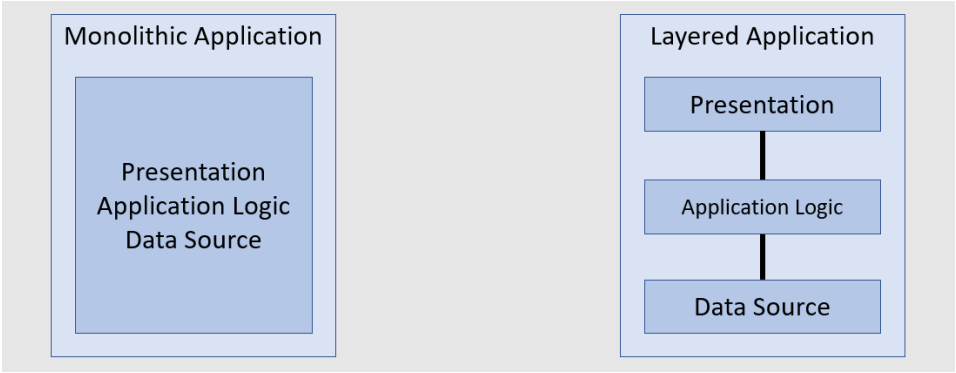

Figure 3.1: Monolithic system design vs Layered system design

	Greenfield	**Brownfield**	**Closed Sorce**
System-of-Systems	Create a new system-of-systems without the need to take legacy into account.	Establish a new system-of-system by creating new APIs and deprecating existing APIs.	Wrap existing systems in a way that creates interface compliance with existing environment.
System	Create a new system to be integrated into existing system-of-systems.	Adapt an existing system such that it can be integrated into a system-of-systems.	An existing system must be integrated in a system-of-systems, but due to a lack of access to its implementation it cannot be adapted. So it must be wrapped in some way.

Table 3.1: Scope and Development Context matrix, table reproduced from Kazman et al. (2013)

will consist of individual systems developed using different technologies, different system design techniques and different standards etc.

It is important to understand the opportunities and limitations the initial state of both individual systems and system-of-systems presents for further development. To do this the scope must be identified, or in other words the existing systems and systems-of-systems must be identified. Then the systems and systems-of-systems are categorized based on the development context (Kazman et al. 2013):

- **Greenfield**: There are no preexisting implementation that restricts the design space
- **Brownfield**: There exists something, but it is possible (in principle) to modify the realization of it
- **Closed Source**: An implementation already exists, but it is impossible to change

Table 3.1 shows how scope and development context are intertwined and define different starting points.

3.3 Important Design Choices

Regardless of whether a system-of-system is developed from scratch or a system is added to a system-of-system important design choices such as how data is shared among the systems and the control regime must be considered. This section presents some of the available options.

3.3.1 Sharing of Data

Data can be common and shared or private and isolated, although somewhere in between these two extremes is most common. When data is shared it occurs either by one-directional information exchange or bi-directional exchange (Kazman et al. 2013).

For a system-of-systems with some level of common and shared data, it is often required to effectively manage access to data that is distributed across multiple repositories. This creates challenges with respect to coordination and the integration architecture will have to cope with challenges like (Microsoft Corporation 2004):

- Multiple records for the same entity in different repositories.
- Semantic dissonance between the data values represented within the same entity across repositories
- The values of information elements might vary over time across parallel instances.
- Violation of referential integrity of data across multiple repositories
- Data synchronization
- Access to logical subsets of data elements that are not available in a single repository

Efficient management of the above mentioned challenges in an integration architecture will normally be handled by introducing an Information Model and implementing a structure for data sharing.

Interoperability

Bass et al. (2013) defines interoperability as the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context. Because a system cannot be interoperable in isolation, specifying the context is important. The context answer questions such as with whom, with what, and under what circumstances? (Bass et al. 2013).

Two or more system can exchange meaningful information only when structural, syntactic and semantic interoperability is achieved. Kazman et al. (2013) defines structural, syntactic and semantic interoperability as follow:

- **Structural**: defined as adherence to relevant standards that describe data exchange.
- **Syntactic**: defined as data exchange occurring with appropriate formats.
- **Semantic**: defined as providing the correct data as part of the data exchange.

When a system-of-systems require involvement of solutions from multiple vendors and organizations information incompatibilities are common. Such incompatibles are costly, and to avoid them efficient utilization of information modelling is needed, or in technical terms interoperability between the systems is needed.

One of the most effective ways to achieve interoperability between systems is to introduce an information model. ”An information model is a technique for specifying the data requirements that are needed within the application domain. It is a representation of concepts, relationships, constraints, rules and operations that specify data semantics for a chosen domain of discourse” (Zhao et al. 2011).

A consistent information model for a system-of-systems that enforces interoperability can be achieved through the following three options (Zhao et al. 2011):

- **The translation approach**: The system-of-systems consist of systems from several vendors. Interoperability is achieved by building and maintaining a data translator that translate from one proprietary format to another.
- **The single vendor mandate approach**: The system-of-systems is supplied by one vendor. Interoperability is achieved as long as the mandate is fully implemented - as long as the systems used and added only is supplied by one vendor.
- **The information exchange standards mandate approach**: The system-of-systems consists of systems from several vendors. Interoperability is achieved by implementing a non-proprietary standard for the information model and only using and adding systems that exchange information in the specified standard.

Usually interoperability is achieved through some blend of these three alternatives. But applying common standards is normally cheaper both to develop and maintain compared to developing and maintain several proprietary formats for the same underlying information. The information exchange standards offers freedom of product choice but depends on conformance and certification definitions and requirements (Zhao et al. 2011).

One-directional Data Exchange

One-directional data exchange is normally used to track the evolution of data in a system-of-system (Kazman et al. 2013). One of the most-used solutions for one-directional data exchange is data warehouse. In systems-of-systems it is often required to track the evolution of data that originates from multiple repositories at the same time, and provide users with a unified view of data from multiple data sources. Delivering a unified view implies the need for the following operations on data from the different sources (El-Sappagh et al. 2011) (Vassiliadis & Simitsis 2009):

- Cleaning of data
- Filtering of data
- Applying simple or complex data validation
- Joining data together from multiple sources
- Transposing rows and columns
- Splitting a column into multiple columns and a row into multiple rows

Also known as Extraction, Transformation and Loading (ETL) operations. These are important activities from an integration architecture perspective to ensure interoperability. The purpose of one-directional data exchange is often to deliver data for analytical and/or decision support. Integrating data across different sources becomes more and more important when the amount of data and number of data sources are growing. Complexity is often related to the analytical part where multidimensional views of the data is requested.

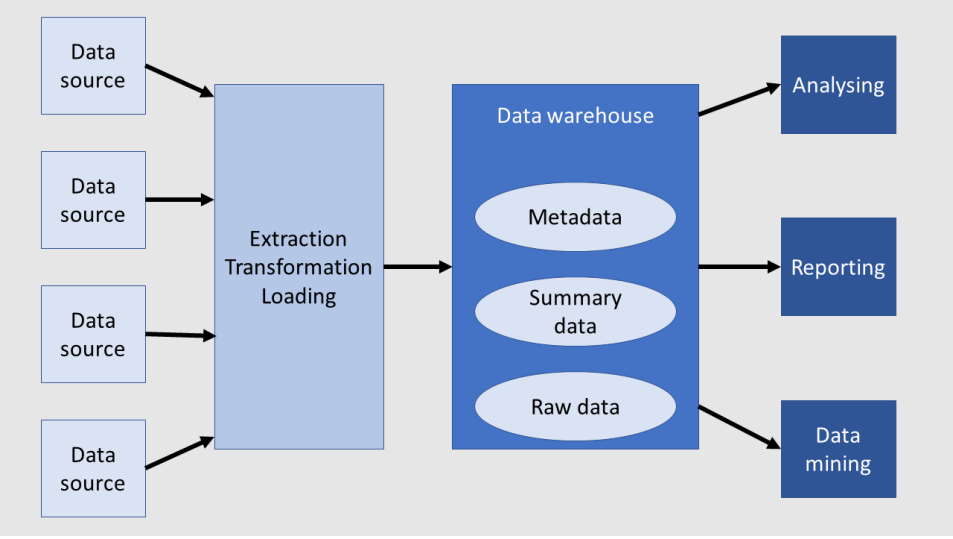

Figure 3.2: Data warehouse

Bi-directional Data Exchange

Bi-directional information exchange refers to situations where two or more systems must exchange information to keep each other in sync (Kazman et al. 2013). This is a fairly complicated procedure, especially in systems-of-systems where data resides in multiple different repositories. By introducing an Entity Aggregation layer, data from different repositories are assembled and presented as one data source to the system. This can only be achieved by

- defining a consistent, unified, system-wide representation of the entity and
- establishing a bi-directional connection between the representation of the entity in the entity aggregation layer and its representative instances in the back-end repository (Microsoft Corporation 2004)

Synchronizing the Entity Aggregation layer with its representative instances in the back-end repository can either be done using straight-through processing or replication. Straight-through processing fetches information from the back-end repositories in real-time and correlates the information into a single view. This implies real-time access to back-end repositories and is also known as online synchronization. Replication, also known as offline synchronization, replicates data from the back-end repositories to a separate physical repository within the Entity Aggregation layer. Replication is required when the following conditions are true:

- Real-time connectivity between the aggregation layer and the back-end repositories is absent.
- To maintain a consistent representation of an entity, complicated joins across multiple instances of the same entity across various repositories is required.
- The solution requires high performance.

Data Representation Usually there are several representation of the same entity in the back-end repositories, therefore it is important to define a system-wide representation of the entity and its attributes and key relationships to other entities. This can be done by either developing custom made entity representations, adopt a representation that is foreign to all the systems in the system-of-system or chose one of the representations supported by one of the existing systems. In large systems-of-system, finding one entity representation might be impossible, due to SOMETHING, then solutions with multiple representations is an option. Even though one system wide entity representation is established, different repositories may hold different schemas for the same entity. To resolve this issue, the entity aggregation layer must take the differences of these schemas into account and apply schema reconciliation.

Data Identification When there are several representations of the same entity, identifying each individual representation, including the representation in the Entity Aggregation

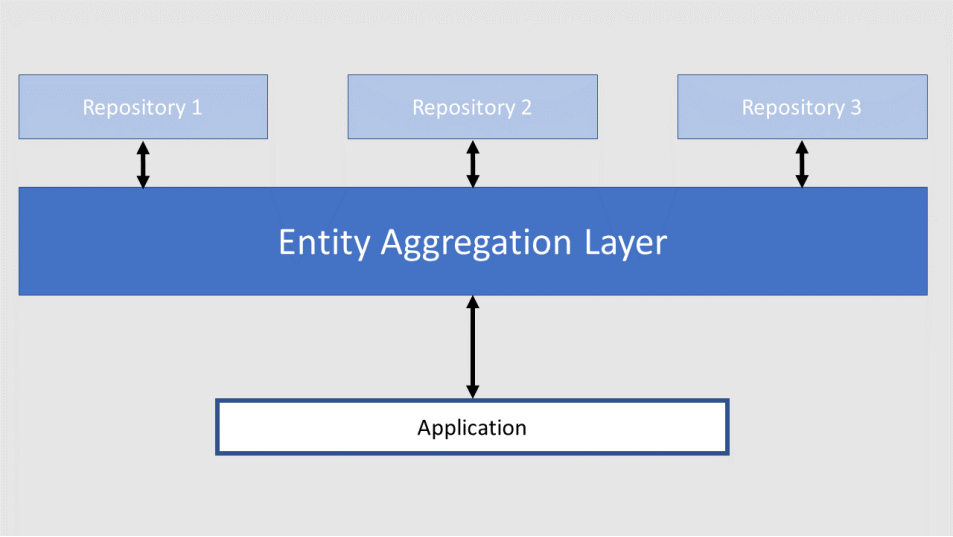

Figure 3.3: Entity Aggregation (Microsoft Corporation 2004)

layer, is not straightforward. By using an entity reference a unique key to the entity instance is created. Both back-end repositories and the aggregation layer needs entity references for their instances of the entity to ensure they have full control over internal data consistency (Microsoft Corporation 2004).

Data Operation During it's lifespan an entity will be subjected to transactional operations such as Create, Read, Update and Delete (CRUD). This leads to challenges related to maintaining the synchrony of data across repositories. To perform inquiries and updates the Entity Aggregation layer uses the entity reference to map to all the repositories. The Entity Aggregation layer also must handle exception in case one of the system fails to go through with the update. In such cases compensation actions must be established. Compensating actions can either be manual or automatic, examples are (Microsoft Corporation 2004)

- Request a rollback on all other updates that have already been made
- Run a compensating transaction to reverse the effects of the successful updates that were already completed.

Data Governance Because different fragments of an entity can stored in different system, establishing an authoritative source for attributes that are represented in more than one system is important. Performing inquiries and updates on a fragmented entity requires different behavior from the Entity Aggregation layer. During inquiries the authoritative source provides the requested attributes, while updates should propagate to all the constituent fragments.

To mange changes across several or all the repositories change management process have to be put in place. The processes are in charge of coordinating changes and ensure that the integrity of the system-wide representation of the entity is not compromised. Changes occurring in the underlying repositories that significantly will affect the Entity Aggregation layer are (Microsoft Corporation 2004):

- Changes in the repository configurations
- Changes to the data model within the repository
- Changes to reference data
- Changes to transactional data in a repository

3.3.2 Control Regime

In a system-of-systems individual transactions and activity sequence will often span multiple systems. The control regime addresses the orchestration and control or the interaction between the involved systems. Activity sequences that span multiple systems can be manged the following ways:

- **Manual control**: The user is in charge of doing the separate tasks in each of the systems.
- **Internal control**: The system initiating the activity sequence directly calls all other systems involved in the execution the activity sequence. This implies implementation of the sequence of transactions into the system initiating the transactions
- **External process manager**: Implement an external process manager that can coordinate the execution of the activity sequence and interacts with the individual applications based upon a predefined process model. External process manager is also know as Process Integration and will be discussed in further detail below.

Process Integration

Process integration involves collaboration between three components, Figure 3.4:

- **Process model**: Contains the overview of the individual steps which constitutes the transaction
- **Process manager**: Creates and manages transaction instances of the process model, maps to functions residing in the different applications, and keeps track of state of the different transaction instances.
- **Application**: Executes specific functions in the process model.

This separates the definition of the process, the execution of the process and the implementation of the individual functions. As a result, each participating application can operate individually without any knowledge of the sequence of steps defined in the process

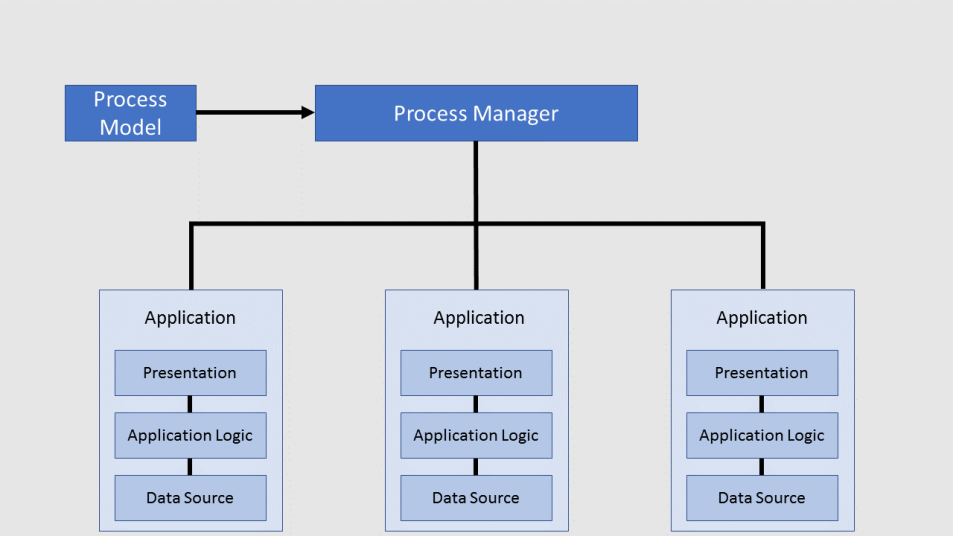

Figure 3.4: Process Integration (Microsoft Corporation 2004)

model (Microsoft Corporation 2004). Another benefit of the separation between model and manager is that the process manager can be domain independent.

Because transactions can take hours or days the process manager typically must be able handle the following:

- **Correlate Message and Process Instances**: Normally several instances of the transaction run concurrently, and messages are sent and received from the different external systems in the activity sequence. These messages are indented for different instances of the parallel transactions. The process manager must be able to identify which messages belong to which instance of the transaction.

- **Transactions**: These transaction can either be Atomic transaction or long-running transactions. Atomic transactions are the same kind of transactions as those found in databases. While long-running transactions are more complicated transactions that typically runs over an extended time period, with other transactions grouped within an transaction and where processes can be stopped and restarted without loosing state.

- **Handling Error and Compensation Transactions**: The most common approach to handle exceptions is to divide the exceptions into status code error and exceptions errors. The status code errors can be handled by conditional logic inside the process model, while exception errors can be handled by exception attached to the scope in which the exception occurred. When an error occur, one of the means to handle it is compensation transactions. Examples of compensation transactions are reverting the transaction state to the state before the error occurred, or use of back up data.

3.4 System Connections

System connection patterns describes different ways to connect systems. This is not straightforward, as existing systems are designed to allow certain types of access and to restrict others. By using the layered system design, three interaction points can be identified, corresponding with the three logic layers: presentation, application and data source. The three main patterns for system connection are:

- Presentation Integration
- Functional Integration
- Data Integration

Figure 3.5 shows a graphical representation of the three main patterns. The three primary patterns can be further divided into derivative patterns, see Figure 3.6. As seen by the figures there are several options when connecting a system with the integration architecture. Sometimes there are more than one way to connect systems. In such situations the trade-offs of each potential choice have to be evaluated.

Although it may seem like there is a one-to-one correlation between the integrating layers and the system connection patterns, that is not the case. A specific integration layer can be used with any system connection pattern. Furthermore, it is possible for a system using one pattern of system connection to communicate with other systems applying other system connection pattern.

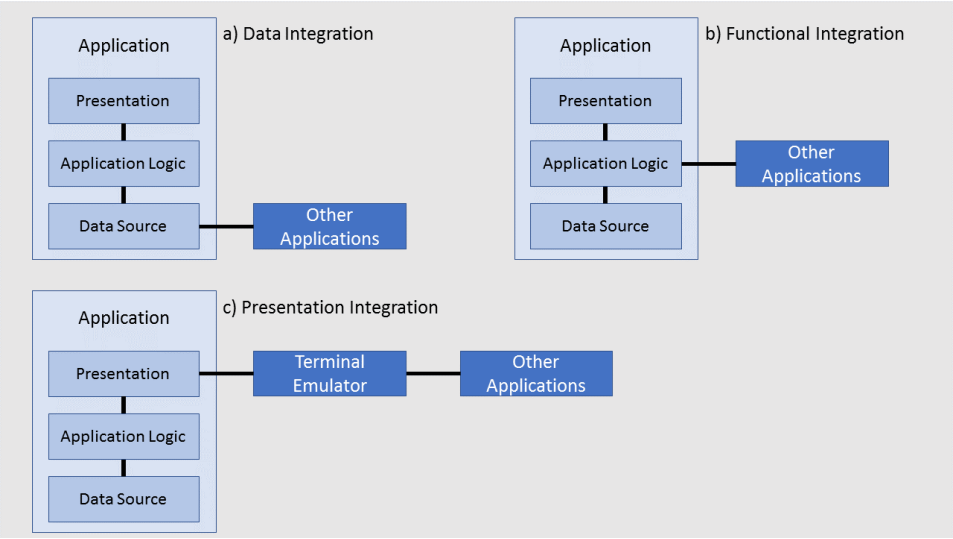

Figure 3.5: The three primary patterns for system connection

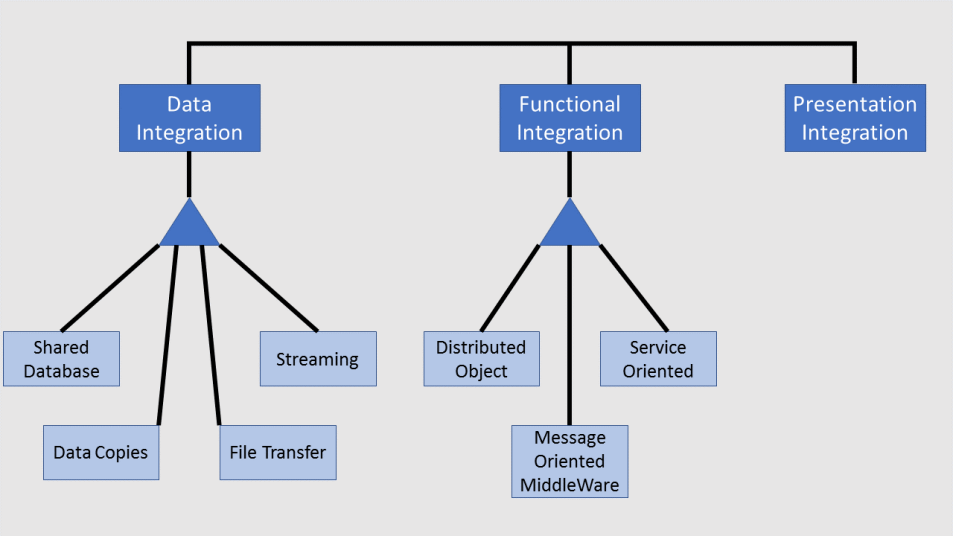

Figure 3.6: Derived system connection patterns (Microsoft Corporation 2004)

3.4.1 Data Integration

Establishing a bridge between two or more applications at the logical data layer is known as Data Integration. In this situation one application plays the role as source application providing target applications with data. The source/target roles are not necessarily a strictly one way relationship. A source application can also be a target application, but for different kinds of data.

Data integration allows applications to access raw data, without going though complicated application logic and validation processes, which is fine as long as there is a ”read only” relationship between the applications. If there is a ”write” relationship between the application some form of validation should be conducted. There are at least four patterns derived from Data Integration; Shared Database, Maintain Data Copies (”Data Copies” in the figure), File Transfer and Streaming. Streaming will not be discussed in any detail, but is used when systems must communicate through continues streams of data, for example video streams (Kazman et al. 2013).

Shared Database In Shared Database several applications share and can access a single data store directly. A Shared Database pattern reduces latency, but will often require the use of a common scheme. Because most databases allows for both reading and writing of data, a transactional integrity mechanism protecting the database from corruption through multiple concurrent updates should be implemented. This will reduce the risk of corrupting the application’s internal state, but cannot protect the database from the insertion of bad data.

Maintain Data Copies While Shared Database uses one instance to store all the data, the Maintain Data Copies makes sure that each application's dedicated data store is always up to date. Maintain Data Copies copy data from one data store to another, basically making several similar instances of the same data stores. This is less intrusive than Shared Databases, but can be a problem due to latency between instances.

File Transfer Files Transfer uses a file to store some piece of data, before transferring it to another application. File Transfer is easy to implement because files are a universal unit of storage for most systems. Latency is a problem and can lead to that the two communication applications losing synchronization with each other.

In general, when implementing Data Integration, latency and synchronization plays an important part, other trade-offs to consider are:

- **Push versus pull**: Should the application pull the data from the data source, or should the data source feed the application when change occur?
- **Granularity**: How much information should be sent at one time? One larger chunk is more efficient than many small, but cohesion between data entities must be understood and accounted for.
- **Master/Subordinate Relationship**: How is data updated? Data can either be updated by one application or by multiple applications. How to keep track of thing if multiple applications update the same data? Can lead to serious synchronization issues.

Deciding on the right kind of data integration pattern to implement is not necessarily straight forward. Factors such as tolerance for stale data, performance, complexity, and platform infrastructure and tool support should be taken into consideration before choosing a pattern. Some of the benefits and liabilities of Data Integration, regardless of derivative pattern, are summarized in Table 3.2.

3.4.2 Functional Integration

In Functional Integration a target application accesses functionality in the source application. This is only possible if the API of the source application can be accessed remotely and the desired functionality exists and are available. Because APIs usually are programming-language specific, there is a need for a middleware interface which works as a translator between the source application and the target applications. The middleware interface converts incoming messages into method invocations of functions that resides in the source application, and converts the return data back into messages that can be transported across the network (Microsoft Corporation 2004). Some of the benefits and liabilities of Functional Integration, regardless of derivative pattern, are summarized in Table 3.3.

Distributed Object Model, Message-Oriented Middleware and Service-Oriented Integrations are the derived from Functional Integration, as seen in Figure 3.6. The patterns can be used individually or combined to achieve the desired connection. When deciding which one to choose, some of the factors that needs to be considered are (Microsoft Corporation 2004):

Data Integration: Benefits	
Non-intrusive	Most databases support transactional multiuser access, ensuring that one user's transaction does not affect another user's transaction.
High bandwidth	Direct database connections are designed to handle large volumes of data
Access to raw data	Raw data tends to be more stable than the transformed data presented to an end user
Metadata	Metadata can aid in the process of transforming from one application's data format to another
Good tool support	Many development and debugging tools are available to aid in connecting to a remote database
Data Integration: Liabilities	
Unpublished schemas	Software vendors reserves the right to make changes to the schema at will
Bypassed application logic	By accessing raw data Data Integration bypasses application logic and validation rules
No encapsulation	The raw data most likely has to be transformed before other applications can use it.
Semantic dissonance	It is difficult to resolve semantic differences between two systems that have slightly different meanings for the same data entity

Table 3.2: Data Integration Benefits and Liabilities, table reproduced from Microsoft Corporation (2004)

- Reliability and latency of the network between endpoints
- Interfaces exposed by your current systems
- Need for interoperability between disparate technical architectures
- Performance
- Fragility, if incompatible updates are introduced to any participant
- Expertise of technical team
- Existing infrastructure

Distributed Object Model Distributed Object Model, also known as instance based collaboration, builds on the object-oriented computing model where objects interact with each other within the application. But instead of local interaction, objects in one application interacts with objects in another remote application. Distributed Object Integration is a viable option if the right infrastructure and expertise are in place. In addition, it depends on fairly reliable and high-speed connection to avoid incompatible updates (Microsoft Corporation 2004).

Message-Oriented Middleware Integration Connection between systems are achieved by using message-oriented middleware to send asynchronous messages. The messages contain small packets of data that are used as a means of communication connecting the systems. In more detail, the target application sends a request to the source application to share some type of functionality using the message queue. The source application takes the message from the queue, interprets it, and processes it. After the source is done processing, it creates a response and send it back to the target using the message queue.

Message-Oriented Middleware Integration uses asynchronous messaging and is therefore a good solution when systems are not reliably connected. The messages will be ”saved” and executed as soon as possible after a system failure or network failure. Messaging is also a way to decouple the sender from the receiver. The disadvantage with Massage-Oriented Middleware Integration is the complexity it imposes.

Service-Oriented Integration A Service-Oriented Integration solution requires a Service Interface and a Service Gateway to expose functionality and encapsulate necessary logic, respectively. Through this set up systems are able to consume and provide XML-based Web-services. Web Service Definition Language (WSDL) contracts are used to describe the interfaces between systems, while SOAP messages are used to enable communication and interaction between systems. By using XML and XML Schema as the basis of message exchange and SOAP as an extensible massage framework a high level of interoperability is reached.

Functional Integration: Benefits	
Flexibility	The abstraction, in the form of a function call, permits many different types of interchanges, such as data replication, shared business functions, or business process integration.
Encapsulation	A functional interface provides an abstraction from an application's internal workings
Robust	Executing a function through the application logic layer ensures that only well-defined functions can be executed, and that data is validated
Familiar programming model	Functional Integration provides a programming model that is more aligned with widespread application programming model.
Functional Integration: Liabilities	
Tighter coupling	One application that sends a command directly to another application results in tighter coupling
Requires application layer to be exposed	Functional Integration is limited to scenarios where the affected applications expose suitable functional interface
Limited to available functions	Limited to functions that are already implemented in the application's application logic
Inefficient with large data sets	Transmitting of large data sets must be done with many individual request, instead of one big one.
Programming-language specific	Many functional interfaces are tied to a specific programming language or technology.

Table 3.3: Functional Integration Benefits and Liabilities, table reproduced from Microsoft Corporation (2004)

3.4.3 Presentation Integration

Presentation Integration is also known as Screen Scraping, a technique that involves the target application collecting information from the user interface of the source application. To do this, the target application simulates user input by using a terminal emulator. The terminal emulator is programmed to translate the actions of the target application to mimic human behavior, Figure 3.5.

Presentation Integration and Web-based interfaces is a good combination because HTML is relatively easy to parse programmatically. On the other hand web-based interfaces are dynamic and are often changed and rearranged. Presentation Integration depends on exact geometric layout of the information, therefore is the Presentation Integration solution very fragile. In Table 3.4 more of the advantages and disadvantages of Presentation Integration are examined.

Presentation Integration: Benefits	
Low risk	All application logic and validation into the application logic protect the internal integrity of the source application's data source.
Non-intrusive	Because other applications appear to be a regular user to the source application, no changes to the source application are required.
Works with monolithic applications	Works well with monolithic applications because it executes the complete application logic regardless of where the logic is located
Presentation Integration: Liabilities	
Brittleness	User interfaces tend to change more frequently than published programming interfaces or database schemas
Limited access to data	Can only access data that is displayed in the user interface
Unstructured information	Little or no matadata attached to the displayed data
Inefficient	Presentation Integration typically goes through a number of unnecessary steps
Slow	Information can be contained in multiple user screens because of limited screen space. Going through multiple screens to obtain information requires multiple requests to the source application.

Table 3.4: Presentation Integration Benefits and Liabilities, table reproduced from (Microsoft Corporation 2004)

Chapter 4

ELMO

ELMO comprises several preexisting software and hardware components: a current profile measurement system with real-time data transmission, a hydrodynamic model, a sedimentation model and a risk assessment tool. By combining the mentioned models, ELMO is able to model the risk imposed on corals by drilling discharges in real-time. The results are presented in a web-based solution allowing for multiple simultaneous users to inspect the results. All the individual models are tied together using a controller application. The solution is implemented in such a way that models and data sources can be exchanged and exist independently (Brönner et al. 2013).

This chapter examines each of the individual systems in ELMO, as well as the front-end application and ELMO-controller. The focus will be on identifying the data needed to compute each of the models, and how the controller orchestrates the information flow between each model and the front-end.

4.1 Hydrodynamic model

The purpose of the hydrodynamic model is to produce hydrodynamic data, that can be passed on to the sedimentation model. There are three ways to obtain hydrodynamic data:

- Use of modelling software - Sinmod
- Use of real-time hydrodynamic measurements
- A combination of the two above mentioned methods - data assimilation.

Sinmod is a fully coupled hydrodynamic and sea ice model, based on the primitive Navier-Stokes equation. Figure 4.1 gives an overview of the different input data used to drive the hydrodynamic model, and produce the ocean and current data. When used as a part of ELMO, Sinmod relies on detailed bathymetry, which usually is provided by the field operator. The boundary conditions are set using a nesting technique, where a large scale model with 20km grid resolution that covers parts of the North Atlantic, the Nordic

Seas and Arctic makes boundary conditions for a smaller grid with 4km resolution. The nesting technique can be used to obtain a resolution of 32m, going through resolutions of 800m and 160m. (Sintef 2017*c*). When large scale models are not available, statistics and tabulated tidal currents are used to produce boundary conditions.

In addition to the boundary conditions tidal components and freshwater inputs needs to be determined, as this has significant impact on the coastal currents. The freshwater input is calculated from seasonal average run off. The freshwater input might affect the salinity level in the water, which together with temperature are two of main water properties. "Wind and air pressure are taken from met.no's data archive. The heat flux is calculated from air temperature, humidity, cloud cover interpolated from available meteorological stations within the model domain and the geological height of the sun" (Slagstad & McClimans 2005).

The real-time hydrodynamic model is obtained by using a hardware solution consisting of a spar buoy with satellite communication system, an Acoustic Doppler Current Profiler (ADCP) and two single-point current meters. The collected data are used to prepare current profiles, and can be used alone as input in the sedimentation model or in combination with Sinmod (Brönner et al. 2013). Data assimilation is used to correct the model state during a simulation. In other words, during a simulation in Sinmod measured real-time data are submitted during the run to correct the model state.

In ELMO the default is to use the assimilated hydrodynamic data. In the current version of ELMO, the Sinmod scenario is set up prior to an ELMO run and stored in ELMO DAP. The scenario is set up only once during an ELMO run, without any subsequent updates. This means that boundary conditions, and water properties are set once during an ELMO run. Tidal input, freshwater input are also set just once, but consists of time series, which will vary with time. The results are produced in NetCDF and stored in ELMO DAP.

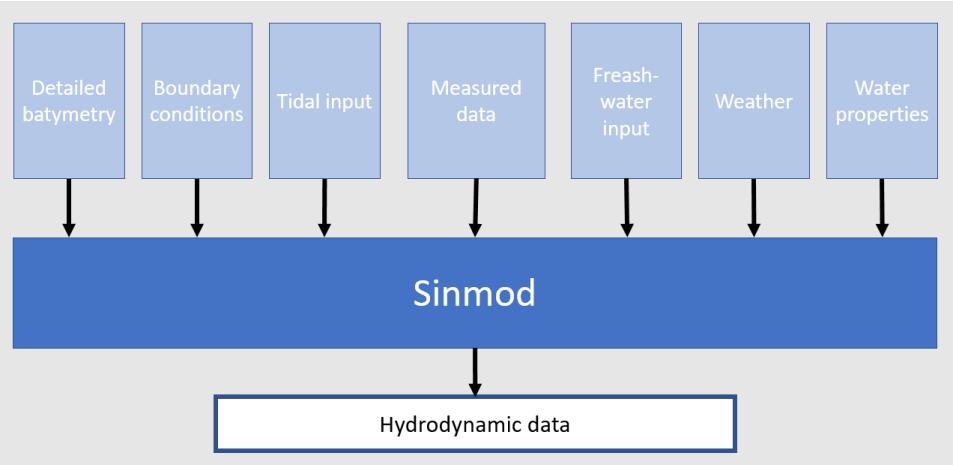

Figure 4.1: Sinmod and associated input data

4.2 Sedimentation model

DREAM is a transport and fate model applying a three-dimensional Lagrangian particle model to track drilling discharges during drilling operations. The model accounts for physio-chemical and transfomative processes such as advection, turbulent diffusion, sinking, dissolution, sedimentation and bio-degradation. DREAM is used to calculate the concentration and mass distribution in the water column and on the sea floor at specified time intervals. (Brönner et al. 2013).

To produce accurate sedimentation data DREAM relies on hydrodynamic data, detailed batytmetry and data from drilling logs and drilling plans, as seen in Figure 4.2. Because the accuracy of DREAM's result to a large extent depends on the quality of the hydrodynamic data, the assimilated data is the preferred input.

The drilling log, drilling plan and batrymetry are provided by the operator. The drilling plan provides characteristics regarding the discharges, such as the the location and depth (below sea surface) of the discharge point. Discharges of cuttings are periodical events, coinciding with the drilling of each section. Different sections require different discharge technology, such as Cutting Transport System, release from platform or close range discharge from top hole. All of the mentioned alternatives dispose drill cuttings at different heights above the sea floor, which will give different results when it comes to sedimentation.

Amounts of cuttings and weighing agents, such as Barite and Bentonite, should be estimated for each section. In addition particle size distribution and density of particle matter present in the discharges should be provided (Rye et al. 2013).

When used as part of ELMO, DREAM's scenario is set up with all the above mentioned parameters and more, before the ELMO run, and stored in locally on the server that runs DREAM. The reason for this is that DREAM is the only system that can read these scenario files. Because it's hard to predict a drilling operation and the amount of discharges produced, it is possible to update the scenario according to the drilling log. This is currently done manually, but a project is initiated to make this process automatic (Nordam 2017). After a run, the sedimentation data are stored in ELMO DAP in NetCDF format.

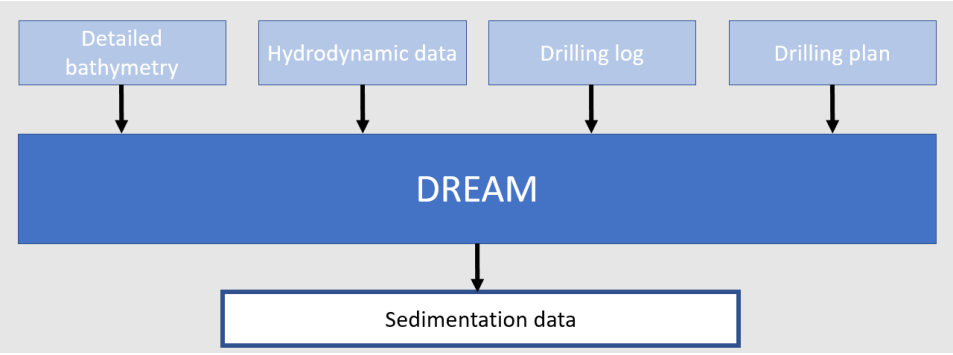

Figure 4.2: Dream and associated input data

4.3 Coral Risk Assessment Tool (C/XRA)

C/XRA is used to quantify the risk imposed on corals (CRA) and other sensitive species (XRA) by drilling discharges. By developing a risk matrix based on threshold values form experiments and the initial state of the corals (dead, damaged, healthy) the potential risk for each coral is calculated. Anchor analysis is not relevant for this project and will be disregarded in further discussions (Ulfsnes et al. 2012).

When applied in ELMO, CRA receives a grid containing information about sedimentation. This grid is produced by DREAM, and is based on drilling plan data and hydrodynamic data. As a result the sedimentation grid represents the drilling plan data, current data and sedimentation data, as shown in Figure 4.3. The map of coral structures and their associated state are handled by DNV GL.

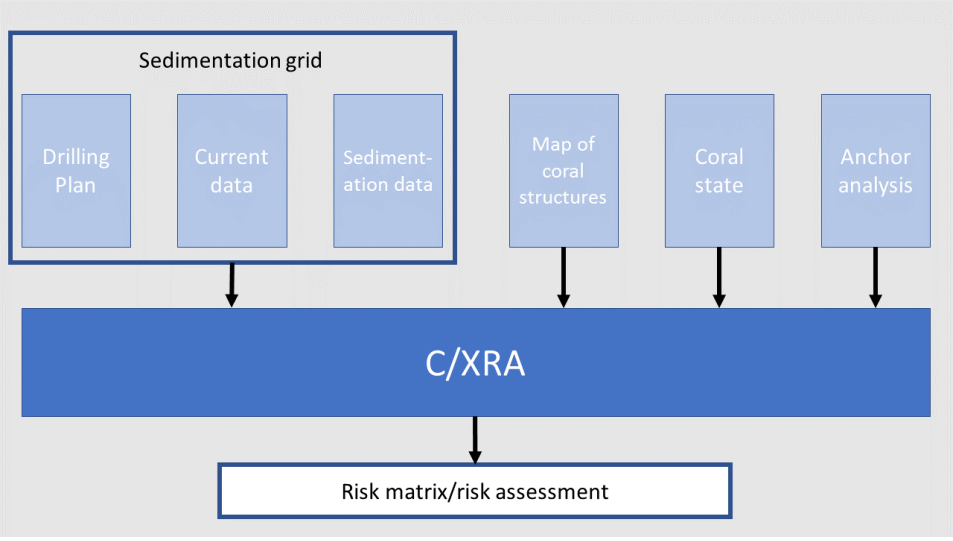

Figure 4.3: C/XRA and associated input data

4.4 Frond-end

The data displayed in the front-end solution can be divided into two categories; static and real-time data, Figure 4.4. The static data comprises bathymetry, coral structures and offshore infrastructure while the real-time data is the current field produced by Sinmod and the real-time measurements, sedimentation from Dream and coral risk from C/XRA. Figure 4.5 is a snapshot of the current ELMO front-end. Since each dataset, static and real-time, is loaded as a single map layer they can be toggeled on and off as desired. There's also a time slider allowing to look at data 3 hours back in time and prediction for 24 hours into the future (Brönner et al. 2016).

The front-end accesses raw data by using a JavaScript library, JsDap, that works as an OPeNDAP client. The data is loaded on JSON format (JavaScript Object Notation), a

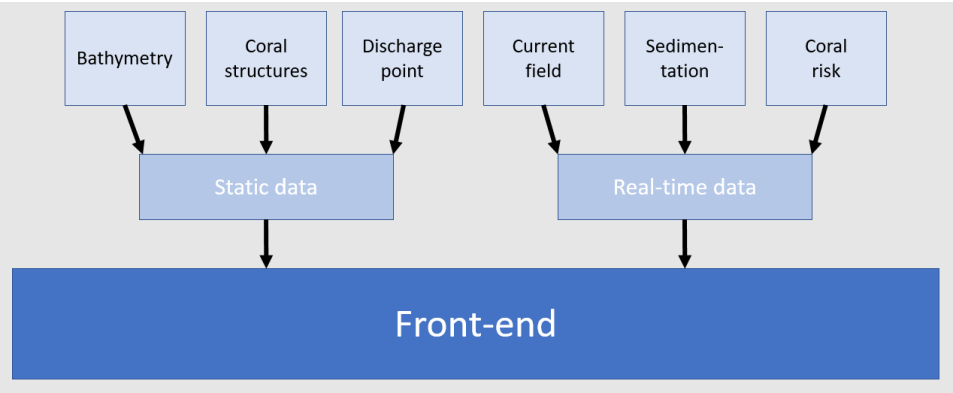

Figure 4.4: Required input data for the front-end application

human readable data-interchange format (json.org 2017). By using GeoJSON, a dialect based on JSON, the data can be easily transformed into map data an be used in mapping platforms like Mapbox and Google Maps (Brönner et al. 2016).

4.5 Controller

ELMO's brain is the control application. Figure 4.6 given an overview of how the controller ties everything together. The control application is in charge of collecting hydrodynamic data from the server, updating the drilling scenario, triggering C/xRA to perform the risk assessment and pushing the result to the front-end application. The controller will begin processing data as soon as new hydrodynamic data is available. Because the systems are independent of each other, only the ELMO controller is aware of all the parts in the system-of-systems (Brönner et al. 2013).

In the current solution all the models and data sources are wrapped using Python applications. Python has powerful libraries for processing NetCDF data, is compatible with ArcGis and support applications where objects can interact with each other over the network, through the Python Remote Objects (Pyro4) library. By using Pyro4 the desired separation between models and data sources is achieved, making every system exchangeable and independent. Each python wrapper is configured for input and output, and code to update the set-up and to run the model or data conversion (Brönner et al. 2016).

ELMO adapts a set of system wide data formats and metadata standards to ensure interoperability. The Network Common Data Form (NetCDF) support storage, access and sharing of grid-oriented scientific data and is a self-describing, machine independent data format. The metatdata standard - Climate and Forecast Metadata Convention (CFMC) - describes what the data represents as well as the spatial and temporal properties of the data. A THREDDS data server is used to access and organise data from the different sources, named ELMO DAP in Figure 4.6. The server is configured to serve NetCDF files via the OPeNDAP protocol. (Brönner et al. 2016).

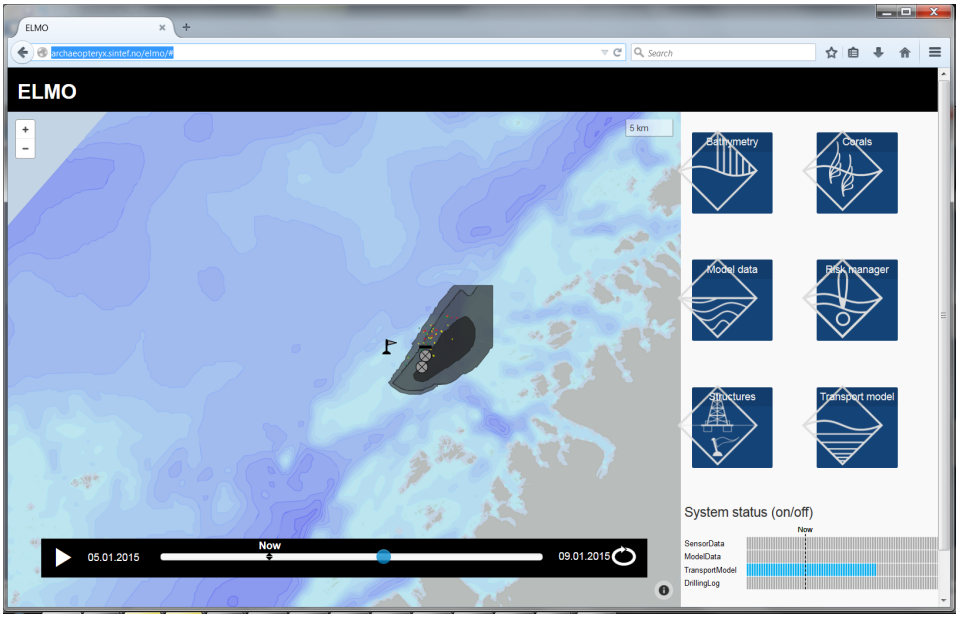

Figure 4.5: Snapshot of current ELMO user interface

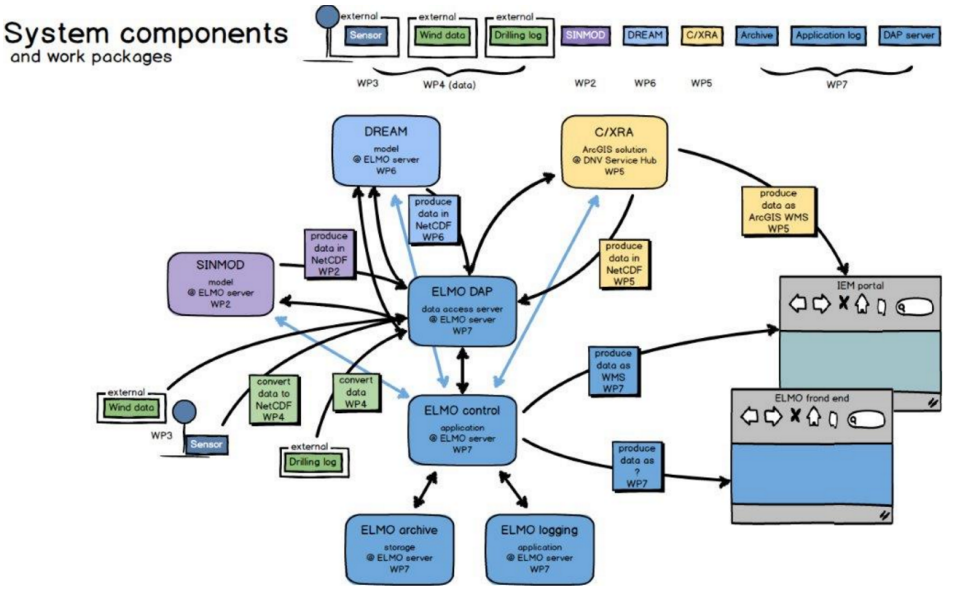

Figure 4.6: Overview of ELMO system components and data flow (Sintef 2017*a*)

Chapter 5

ELMO - Integration Architecture Analysis

This chapter presents the analysis of ELMO's integration architecture. The analysis follows the framework presented in chapter 3 and is based on the information presented in chapter 4.

5.1 System Design

Neither Sinmod, DREAM or C/XRA are designed to participate in a system-of-systems. As a result there are no available APIs which can be accessed by the ELMO controller application in the Data Source layer or Application logic layer. Communication between the different systems and the controller are strictly limited to the Presentation logic layer.

To deal with this limitation, all systems have been modified to access and store data using NetCDF, where this was not already the case. Python and shell scripts was used to wrap Sinmod and DREAM, providing a way to modify input data and start simulation runs.

5.2 Development Context

Although there are several stakeholders in the ELMO project, Sintef is the partner responsible for developing the ELMO software solution. This is reflected in the division of responsibilities for the various systems. All systems, with the exception C/XRA and the front-end application, which was outsourced, are developed by different departments in Sintef. Table 5.1 summarizes the scope and development context of all the individual software systems in ELMO and ELMO as a whole.

DREAM was adapted to read input and produce output in NetCDF, and an interface for updating drilling scenario was implemented with functionality that allows restarting of

simulations from stored states. Sinmod underwent a process to improve data assimilation with a variety of input sources and produce GIS compatible output data (Brönner et al. 2015). Both Sinmod and DREAM existed prior to the ELMO project, which makes both of them Brownfield systems.

As a rule of thumb, outsourced systems are harder to adapt than in-house systems. But to comply with the need for real-time risk assessments, the CRA module had to be operationalised, to retrieve data and produce risk assessments regularly (Brönner et al. 2016). Which means that C/XRA is a brownfield system. The Front-end application and the ELMO controller did not exist prior to the ELMO project, and were both customized to fit the already existing systems. Giving both systems a greenfield status.

As mentioned, several of the systems existed prior to the ELMO project which makes ELMO either a brownfield system-of-systems of a closed source system-of-systems. Due to the lack of available APIs in Sinmod, DREAM and C/XRA, ELMO is forced to use wrappers to access and control the systems, which makes ELMO a closed source system-of-systems.

Name	Scope	Development Context	Responsible
Sinmod	System	Brownfield	Sintef
DREAM	System	Brownfield	Sintef
C/XRA	System	Brownfield	DNV GL
Front-end	System	Greenfield	Sintef - Out-sourced
Controller	System	Greenfield	Sintef
ELMO	System-of-systems	Closed Source	Sintef

Table 5.1: Summary of Scope and Development context for ELMO

5.3 Sharing of Data

Table 5.2 gives an overview of all the data sources used in ELMO. Even though all data sources are categorised as shared, this is a truth with modifications. Some of the data sources aren't available from ELMO DAP, but are necessary to set up Sinmod's or DREAMS's scenario and is therefore categorized as shared, because they are shared within some instance of ELMO.

For several systems to be able to share data, interoperability is a necessary prerequisite. This section will discuss the methods used to ensure seamless sharing of data in ELMO.

5.3.1 Interoperability

Two approaches are used to achieve interoperability in ELMO. Internally ELMO applies an information exchange standard. The information exchange standard is the Climate and Forecast Metadata Convention (CFMC) for NetCDF files (CF-NetCDF). CFMC creates an internal information model used to describe all entities in ELMO, with the exception of drilling log data.

System	Data Source	Data provider	Static vs Dynamic	Shared vs Private	One-directional vs Bi-directional
Sinmod	Bathymetry	Operator	Static	Shared	One-directional
	Boundary conditions	NMI Ocean model	Static	Shared	One-directional
	Tidal components	Statistics - research	Static	Shared	One-directional
	Freshwater input	Statistics - research	Static	Shared	One-directional
	Weather	NMI	Dynamic	Shared	One-directional
	Water properties	NODC World Ocean Atlas	Static	Shared	One-directional
DREAM	Bathymetry	Operator	Static	Shared	One-directional
	Hydrodynamic data	Sinmod	Dynamic	Shared	One-directional
	Drilling log	Operator	Dynamic	Shared	One-directional
	Drilling plan	Operator	Static	Shared	One-directional
C/XRA	Sedimentation gird	DREAM	Dynamic	Shared	One-directional
	Coral structures	DNV GL	Static	Shared	One-directional
	Coral state	DNV GL	Static	Shared	One-directional
ELMO	Wind	NMI	Dynamic	Shared	One-directional
	RT hydrodynamic data	Buoy	Dynamic	Shared	One-directional
	Drilling log	Operator	Dynamic	Shared	One-directional
	Sedimentation data	DREAM	Dynamic	Shared	One-directional
	Risk assessments	C/XRA	Dynamic	Shared	One-directional

Table 5.2: Overview of all data sources in ELMO

Incorporating drilling log data is an ongoing project, and it is expected that WITSML will be used as a standard together with a wrapper which can translate to DREAM's proprietary format (Nordam 2017). Figure 5.1 gives an overview of how the different systems and metadata standards are intertwined.

To ensure that external data comply with the internal information model, all entering data are translated. This means that all data relevant for CFMC is translated from local metadata into Climate and Forecast metadata, and all data formats are translated from local format to NetCDF format. As a result, structural, syntactic and semantic interoperability is achieved in ELMO.

Selecting standards for interoperability is a very important and strategic decision within a system-of-systems. Changing interoperability standards in system-of-systems where systems share many different data entities, have wide-range impact and normally implies significant changes to the overall solution.

In ELMO the selection of the Climate and Forecast Metadata Convention seems to be a good choice. The CFMC information model has matured over a long period, introducing several versions, and have become the de-facto standard within the ocean science community (Brönner et al. 2015). This gives reason to believe that CFMC can be a stable element in ELMO, and as such costly adaptions to new standards can be avoided in the future.

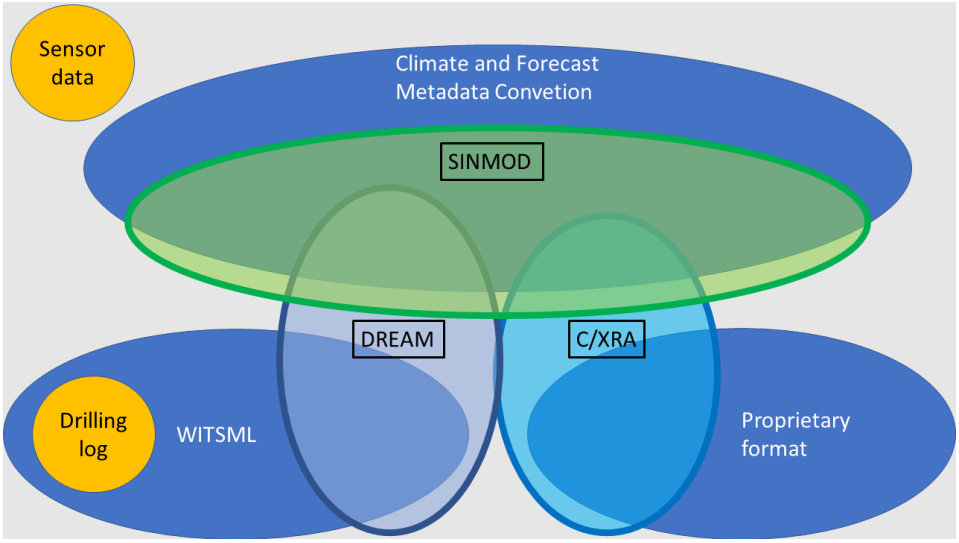

Figure 5.1: Relationship between information models and individual systems in ELMO

5.3.2 Data sharing

ELMO applies one-directional data exchange. To produce an accurate result ELMO relies on data from multiple data sources. Figure 5.2 show how data travels through the different instances in ELMO. For each step, value is added, before the data is presented to the end user. This requires comprehensive integration of data on all levels.

By having a closer look at Sinmod, this process can be more thoroughly investigated. Figure 5.3 gives an overview of how data is transformed from relatively basic information to more profuse information. By combining weather, boundary conditions and more Sinmod is able to produce detailed hydrodynamic data. In addition Sinmod is able to make corrections to the hydrodynamic data if the input contains measured current data, ensuring even more accurate hydrodynamic data. It is assumed that similar processes occur in DREAM and C/XRA as well.

Combing all data sources, including data sources that are native to only one of the systems, ELMO makes use of a wide range of data sources as seen in Table 5.2. To be able to preserve interoperability within ELMO and within the different systems several Extraction, Transformation and Loading (ETL) operations are necessary between the system and data sources. If any of the systems uses proprietary data formats and metadata standards, the ELT processes become even more crucial.

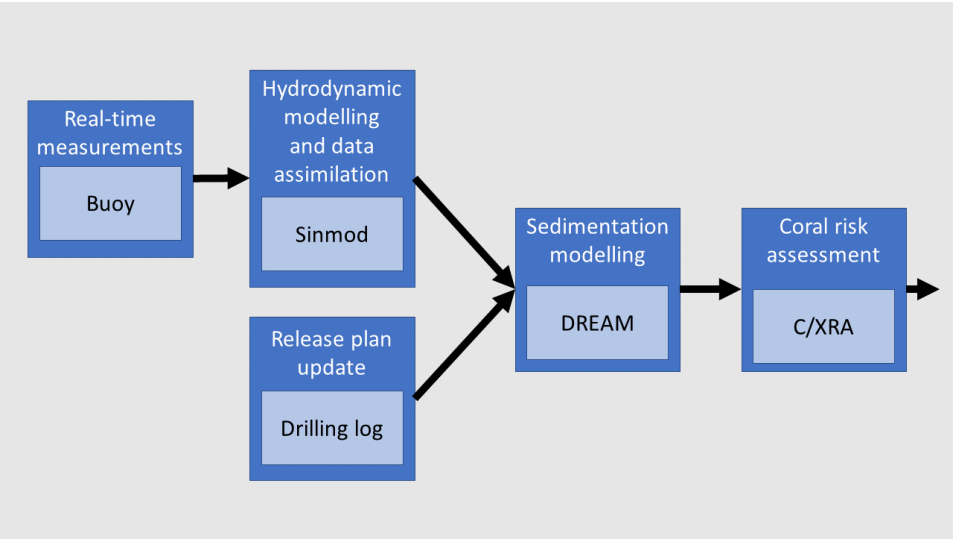

Figure 5.2: Data flow and aggregation through ELMO (Ulfsnes et al. 2014)

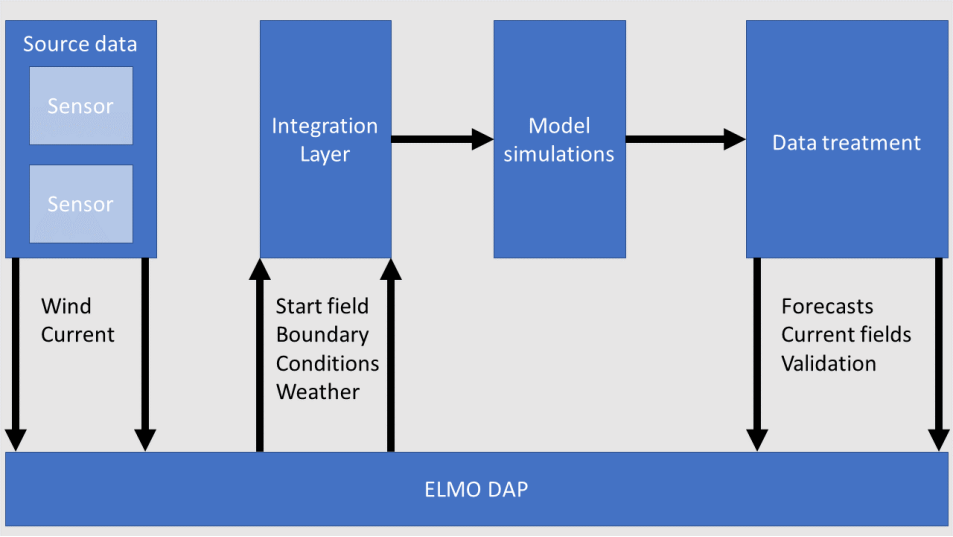

Figure 5.3: Data flow and aggregation through Sinmod (Brönner et al. 2013)

5.4 Control Regime

It is believed that ELMO applies an internal control regime, where the controller undertakes the role as initiating system. The initial scenarios are set up and stored manually on the server(s) running the ELMO system, which implies some level of manual control as well. The activity sequence once an ELMO scenario is up and running consists of the following steps:

1. Collecting and verifying hydrodynamic data from sensors
2. Run hydrodynamic model (Sinmod)
3. Update of hydrodynamic data, including archiving and logging
4. [Update DREAM scenario]
5. Run sedimentation model (DREAM)
6. Update of sedimentation data, including archiving and logging
7. Run risk assessment model (C/XRA)
8. Update of risk assessment data, including archiving and logging
9. Push updated data to front-end application

During the run, hydrodynamic data is collected and updated at regular intervals, leaving the initial Sinmod scenario untouched. The hydrodynamic data is passed on to DREAM, and used to calculate the sedimentation. Because drilling operations are unpredictable it

is possible to update DREAM's scenario during the run in case of some unexpected event. The brackets indicate that updates doesn't necessarily take place for every time step. The last step before updating the user interface performs the risk assessment based on the sedimentation grid produced by DREAM.

The control regime used in ELMO is very basic, and is mostly concerned with data flow, and executing the steps in the order required due to dependencies. This is a good solution for an environment of ELMO's current size. In the future, ELMO is meant to monitor multiple drilling operations which will lead to a need of a more advanced control regime. Controlling several parallel processes requires more advanced exception handling, more check points, SOMETHING, and SOMETHING, to mention a few examples. Which means it might be beneficial to go from an internal process regime to an external process regime, also known as process integration.

5.5 System Connections

As mentioned previously the only way to establish a connection between Sinmod, DREAM, C/XRA and ELMO is to apply Presentation Integration. Presentation integration requires a terminal emulator to translate commands from the controller into what is perceived as human actions by the individual systems. This translation is conducted by the python wrappers associated with each system, see Figure 5.4. This enables the controller to access Sinmod, DREAM and C/XRA and start the individual simulations in accordance with the activity sequences presented in the Control Regime.

ELMO DAP is a THREDDS data server, and works as a shared database storing data that is common for Sinmod, DREAM and C/XRA. A shared database solution requires a common schema, and mechanisms to prevent semantic dissonance. This is resolved by using the Climate and Forecast Metadata Conventions and NetCDF. As mentioned previously, applying the mentioned standards ensures structural, syntactic, and semantic interoperability between all systems in ELMO. Database connections are designed to handle large volumes of data, which is necessary as the files containing the hydrodynamic data and sedimentation data are assumed to be of considerable size.

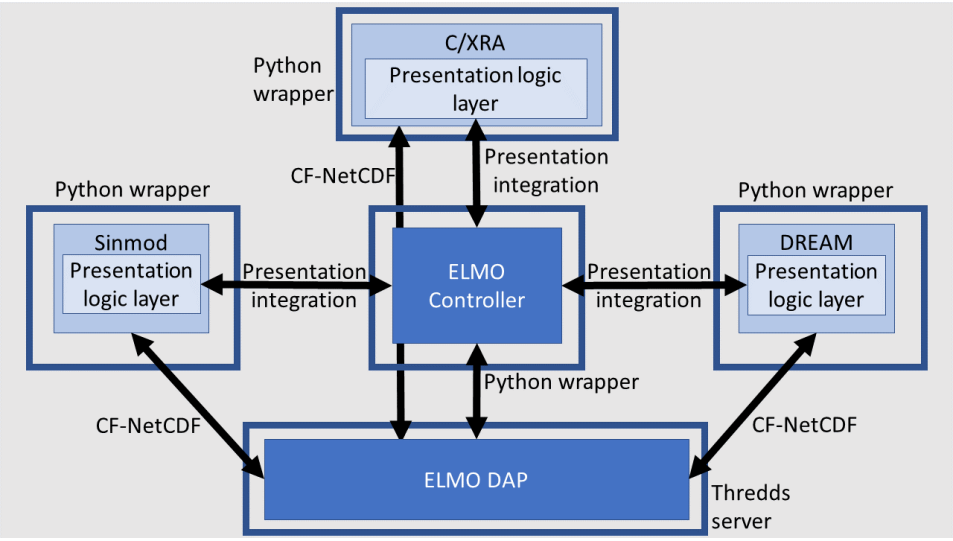

Figure 5.4: System connections in ELMO

5.6 Conclusion

By combining the functionality of Sinmod, DREAM and C/XRA, ELMO is able to present the end users with new functionality that improves the opportunity for environmental monitoring. All of the above mentioned systems were developed as standalone systems based on proprietary information models, with little or no standard integration capabilities.

Based on the analysis the following can be concluded: The key elements in ELMO's integration architecture are a solid information model and a shared database to store common data. Because data exchange only occurs in one direction, an internal control regime, implemented in the ELMO controller, is sufficient to handle the data flow and the execution of the activity sequence. Presentation integration is used between the ELMO controller and the systems user interfaces to initiate the runs of the individual systems. ELMO's complexity is assumed not to be related to the the integration architecture but lies in the mathematical models, used to produce the hydrodynamic data and sedimentation data.

Chapter 6

Front-end Prototype

In conversations with Statoil and Sintef it was revealed that additional front-end functionality in ELMO was desired. The key question was, how can we increase the value of the data available in ELMO? Several options was discussed, but a feature that displays sedimentation at a point-of-interest (coral) in the vicinity of a discharge point was the most feasible one. A rough picture of the desired output was painted, but left room for individual adaptions. Therefore, a feature informing the user about the progress of the drilling operation was added.

Inflicting any non-reversible damage to areas surrounding drilling operations is illegal, and the consequences are severe for both companies (large fines) and nature. The above mentioned functionality can be used as a tool to prevent damage by increasing the knowledge regarding the state of corals, and by indicating when necessary precautions must be taken.

The main purposes of the practical project was to acquire the necessary competence and to be able to present the relevant data in a functioning front-end solution. Appearance and detailed design has been given little attentions as it's not crucial to complete the task.

This chapter is divided into four sections. The first section present the target environment, the second section examines the prototype solution, before a analysis is conducted in section 3 and a conclusion is drawn in section 4. This chapter will to some extent revolve around references to functions, array and variables. Functions and arrays are denoted by parentheses () and square brackets [], respectively. While variables are plainly written out with their respective variable names, such as areaElementId.

6.1 Target Environment

The target environment describes the target front-end solution. In other words, the resulting front-end solution given unlimited access to data sources, documentation, programming experience and the ELMO solution. First, this section gives an overview of the main features of the target solution. Then, it examines the underlying data displayed in the features. In the third subsection the functionality is presented through the use of user actions.

6.1.1 Front-end features

Figure 6.1 shows a sketch of the layout and design of the target front-end solution. The target solution consist of eight elements assigned the letter A to H. The drop down menu (A) is a list of active drilling operations with their associated discharge points. B is an interactive map that enables the user to choose a point-of-interest (poi) for inspection. C is a table containing information regarding the drilling operations. Although it only currently shows start date, duration and estimated end date other relevant information could be added. C is meant as a supplementing feature to H.

The exact location of the point-of-interest, indicated by the white cross on the map, can be viewed in E. F displays the accumulated sedimentation on the point-of-interest. The color of the graph indicates the risk imposed on the coral according to the risk table (D). H is the progress chart that presents the planned progress at a given time compared to the real-time progress. The last feature is the time slider, which enables the user to go back in time to the start of the operation.

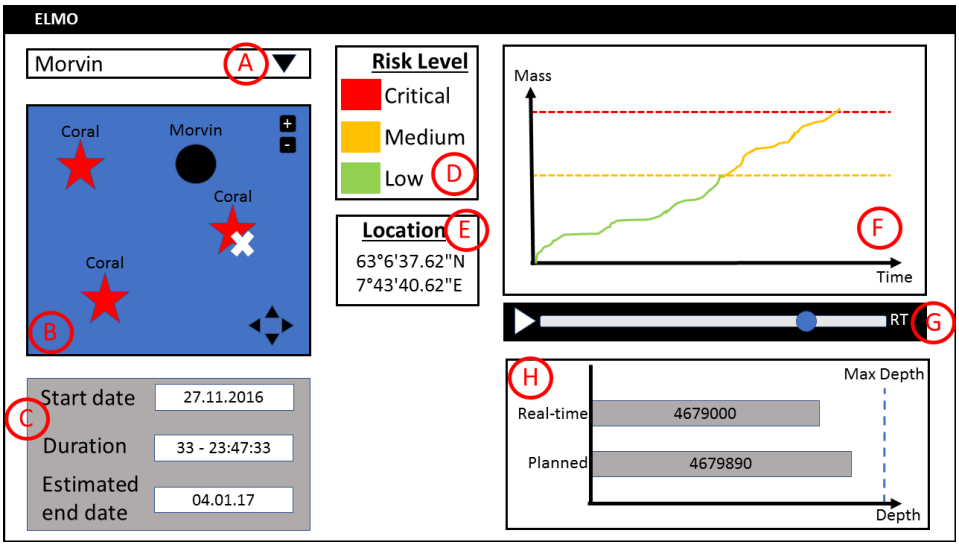

Figure 6.1: Layout and design of the target solution

6.1.2 Data Sources

The data sources needed to display all the above mentioned feature can be viewed in Figure 6.2. The data sources are divided into static and real-time data. Although the drilling plan may be updated during a drilling operation, it is still categorized as a static data source because the rate of update is relatively slow compared to the real-time discharge data and sedimentation data (approximately every hour). The relationship between the data sources to the front-end features can be viewed in table 6.1.

As mentioned in Chapter 2 the drilling plan contains information about the drilling operation. The drilling plan provides the target solution with information regarding the

Front-end feature	Data Source
(A) Drop down menu	Active drilling operations
(B) Map	Bathymetry map, coral structures, discharge point
(C) Drilling info table	Drilling information, time
(D) Location	Derived from the map
(E) Risk level table	Risk threshold values
(F) Progress chart	Real-time discharge, planned discharge
(G) Time slider	Drilling information and time
(H) Accumulation chart	Sedimentation, risk threshold values

Table 6.1: Relationship between front-end feature and data source

discharge point (coordinates), the planned release and general drilling information such as start date, estimated end date and so on. Time is mentioned as a data source in Table 6.1, but not in Figure 6.2, the reason behind this is that time isn't an external data source. The time data source is the Unix Time which is a "universal" data source, and can be accessed from almost all computers. As a result duration can be calculated by subtracting the start date from "Time".

Active drilling operations, bathymetry and the real-time discharge are provided by the operating company, while sedimentation is provided by DREAM. Prior to a drilling operation the locations of coral structures can be known or unknown. If the location is unknown the area surrounding the wellsite needs to be mapped. Then the known locations are displayed in the map feature. The risk threshold values are based on research (Ulfsnes et al. 2012) and are calculated for each individual coral, based on the initial state of the coral.

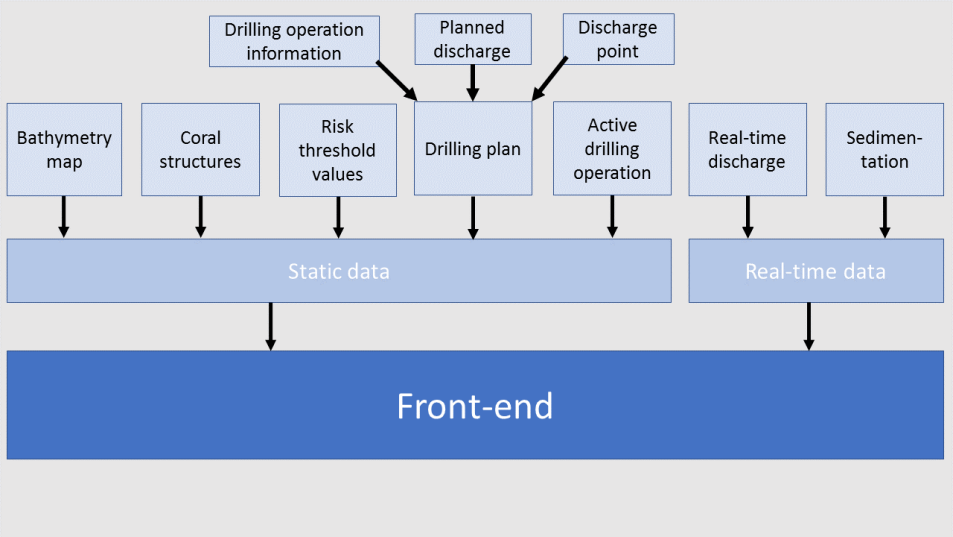

Figure 6.2: Required data sources for the target solution

6.1.3 Functionality

The front-end functionality can be divided into four user actions. In Figure 6.3 and 6.4 the user actions and their respective response and associated data sources can be viewed. These user actions have to be conducted successively, in order to get the desired output pictured in Figure 6.1.

The first user action is ”Page loaded”. This action loads the web page interface. Most of the features will be empty with the exception of the drop down list containing the discharge points and the map. The map will be fully zoomed out, displaying the coast of Norway, the North Sea, Norwegian Sea and the Barents Sea. Although its possible to select the point-of-interest already now, it’s not recommended.

Therefore, to make it easier to inspect the right coral, the second user action is ”Discharge point selected”. This user action have three responses. First, the map is updated, with a zoomed view of the area surrounding the discharge point including any sensitive fauna. Second, information regarding the drilling operation is displayed, and third the planned and real-time progress of the drilling operation displayed.

The next user action is ”Point-of-interest selected”, which means the user have selected a coral for further inspection. The location of this coral will appear in the location table, while a line graph indicating the accumulated sedimentation with the associated risk is displayed. The last user action is ”Time changed”. This action makes it possible for the user to go backwards in time, all the way back to the start of the drilling operation. Both the sedimentation chart and progress chart will update according to the specified time.

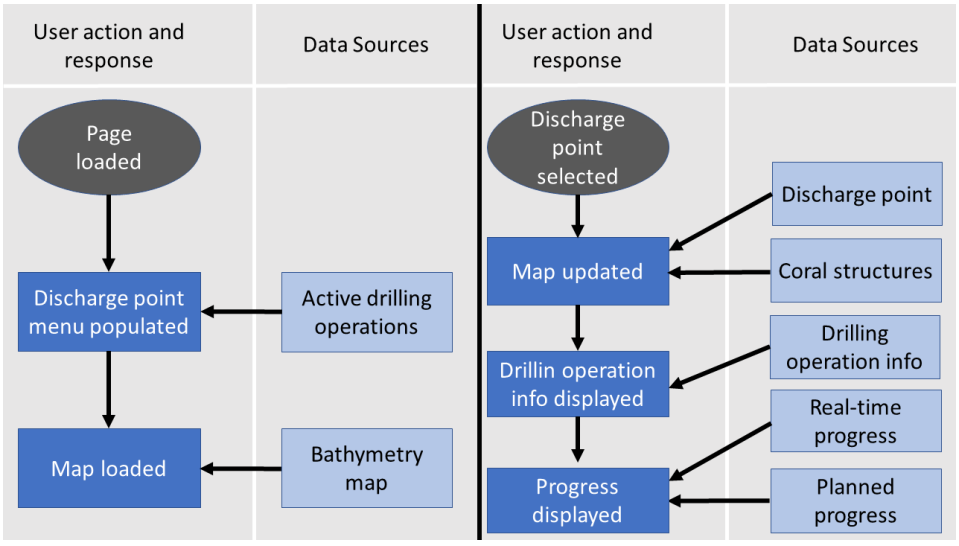

Figure 6.3: Target user actions and response: ”Load Page” and ”Select Discharge Point”

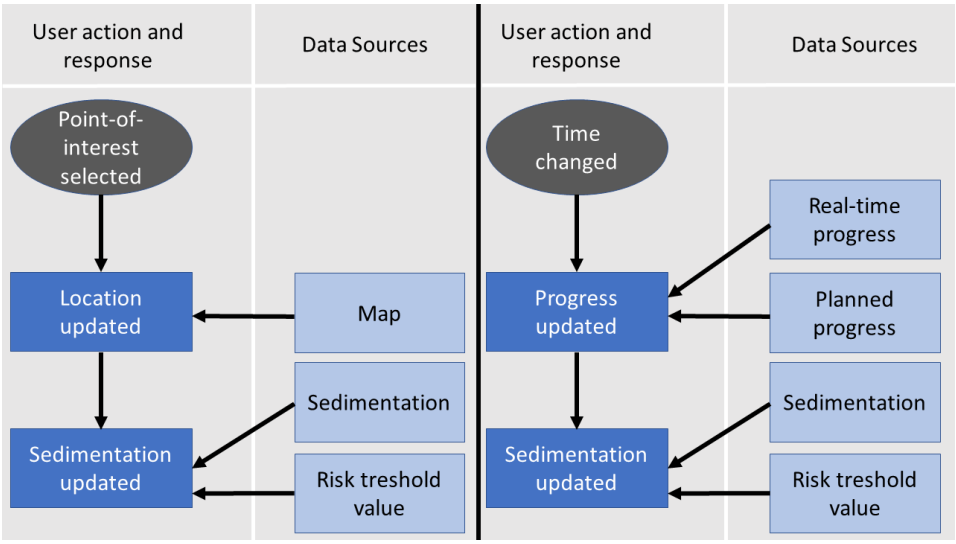

Figure 6.4: Target user actions and response: ”Select Point of Interest” and ”Change Time”

6.2 Prototype Environment

This section discusses the front-end prototype, the practical task of this master's thesis. First this section elaborates on the framework used to implement the front-end prototype, and the assumptions and restrictions for the prototype. Then this section presents the front-end feature, data sources and functionality of the prototype solution, before conducting an analysis where the prototype is compared to the target solution and the structure and technical aspects of the results are discussed.

HTML5, CSS and JavaScript is used to develop the foundation of the front-end solution. To obtain an expedient user interface, with charts and the functionality needed D3.js (D3), a JavaScript library is used. D3 (Data Driven Documents) allows the user to bind arbitrary data to a Document Object Model (DOM), an then apply data-driven transformation to the documents (Bostock 2017).

D3 have been used to make the accumulation chart and progress chart, which would have been much more troublesome without D3. Although none of the data sets used in the prototype were of any significant size, D3 supports large data sets with minimal overhead and is also open for dynamic behaviours for interaction and animation.

The whole prototype is developed locally with a relatively basic setup. JsDap is used to access the Norwegian Meteorological Institute's THREDDS server. Because most browsers don't allow downloading data from external domains using JavaScript, the Allow-Control-Allow-Origin Google Chrome plug-in is used as a quick fix to solve this problem.

6.2.1 Front-end features

Figure 6.5 shows the complete layout and design of the prototype. The six features in the prototype are denoted A to F. A is a drop down menu consisting of a list of active drilling operations with their respective discharge points. Instead of using a map to display the discharge point and the surrounding area a static image (B) is used. In the image, points-of-interest, such as corals, are identified with labels. These labels are listed in the second drop down menu (C), so that the user can choose which coral to investigate closer. The accumulation chart is indicated by D, while the progress chart is denoted by E. There are two buttons under the progress chart (F), enabling the user to go back and forth in time as desired.

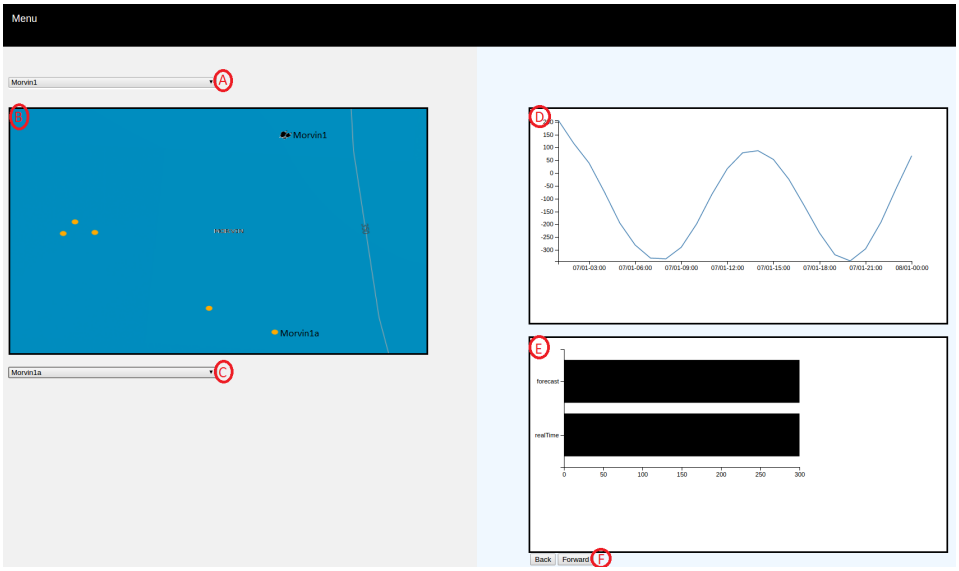

Figure 6.5: Layout and design of prototype solution

6.2.2 Data Sources

Because of limited access to real-time and real-life data, most of the data used in this prototype are ”dummy data” - made up data used to test the functionality. Figure 6.6 gives an overview of the data sources needed to display the front-end prototype, and their associated dummy data. The content of the dummy data is summarized in Table 6.2.

Each element in both drop down menus consist of a name-value pair. The element names are displayed in the drop down menu and the values are used as parameters in JavaScript functions enabling a dynamic front-end. For instance, wellboreArray[] is used to store the element names in the wellbore drop down menu, while wellboreValueArray[] contains the values, which are file names of the images representing each wellsite.

The name and value of the elements in the coral drop down menu are stored in coralArray[] and coralValueArray[], respectively. But instead of linking to locally stored data, the

values in coralValueArray[] are URLs linking to the Norwegian Meteorological Institute's THREDDS server. These links specifies the location, time interval and what kind of data should be sent in return. In this case the JSON response contains information concerning the current velocity at a specified location in the Norkyst800 data set.

There are two main reasons for using this data. First, the Norwegian Meteorological Institute uses a THREDDS server, the same server system used in ELMO. As a result, the function and procedures used to access and parse the data can be used in the future with only minor adaptions. Second, the data are open to the public, therefore no special access is needed.

datasetDepth[] is a lightweight JSON array containing two JSON objects; one object for the planned depth (forecast) and one object for the real-time depth (realTime). The decision to implement datasetDepth[] as a JSON array was based on the fact that it's a frequently used data format and some level of knowledge regarding JSON arrays and objects is handy.

Array	**Contents**
wellboreArray	Array of active drilling operations
wellboreValueArray	Array of image names
coralArray	Array of coral names
coralValueArray	Array of links to NMI's THREDDS server
datasetDepth	Array with two JSON objects: forecast and realTime

Table 6.2: Overview of data sources in prototype solution

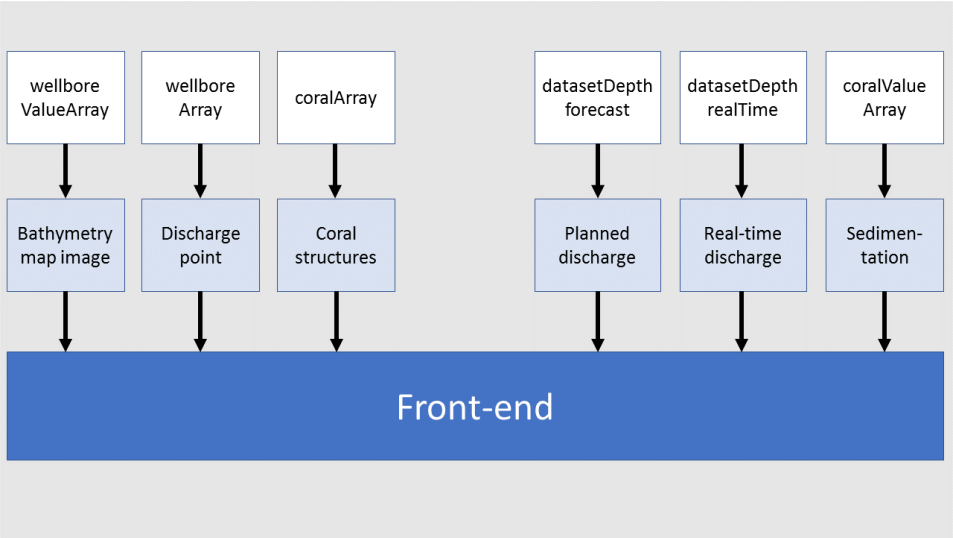

Figure 6.6: Data sources used in prototype solution

6.2.3 Functionality

The prototype environment consist of four user actions; Page loaded, Wellbore selected, Coral selected and Time changed (Forwards/Backwards). How these user actions are connected with function calls and their respective data sources can be viewed in Figure 6.7 to 6.10.

Page loaded The immediate response, when the page is loaded, is the population of the wellbore drop down menu. The wellbore drop down menu uses wellboreArray[] and wellboreValueArray[] to populate each element in the drop down menu with name-value pairs.

Wellbore selected When the user selects a wellbore, loadImage() and populateCoral() are triggered by the change in wellbore drop down menu. loadImage() and populateCoral() uses the areaElementId to load the image and coral elements corresponding to the selected wellbore, while coralElementId specifies where these coral elements are to be added in the front-end.

Coral selected Selecting a coral in the coral drop down list for further inspections triggers a series of function. First loadUrl() is called, which in turn calls loadData(), both of which will be thoroughly discussed later in this chapter. loadData() then in turn calls createLineGraph() and createBarChart(). createLineGraph() renders the accumulation chart based on the input using D3 functionality. When the progress chart is rendered for the first time, the input parameter in createLineGraph() is zero (0), indicating that the last available data should be displayed. createLineGraph() calculates the id number of the last element and passes it to drawRectangles(), which will be discussed later together with loadUrl() and loadData().

Forwards or Backwards When the forwards or backwards buttons are pressed, the progress chart is updated. The procedure is the same as described above, but instead of zero (0) being the input parameter in createBarchart(), minus one (-1) is used for backwards and one (1) is used for forwards. If the forward button is pressed, when the front-end is in ”real-time mode”, an error message alerts the user that it is not possible to go to the next time frame. The same applies if the user tries to go further back in time than possible.

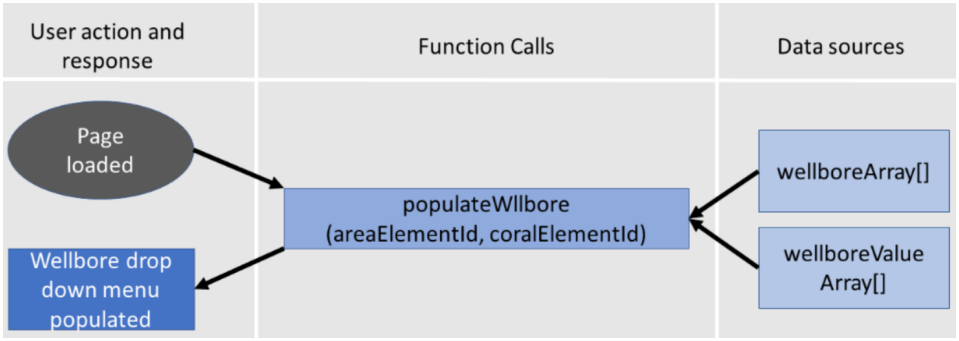

Figure 6.7: Page loaded user action and response

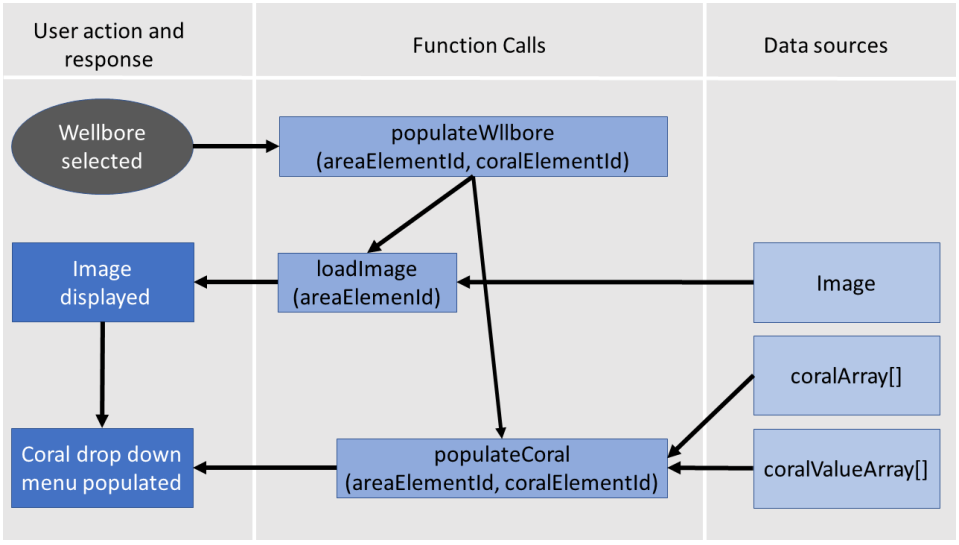

Figure 6.8: Wellbore selected user action and response

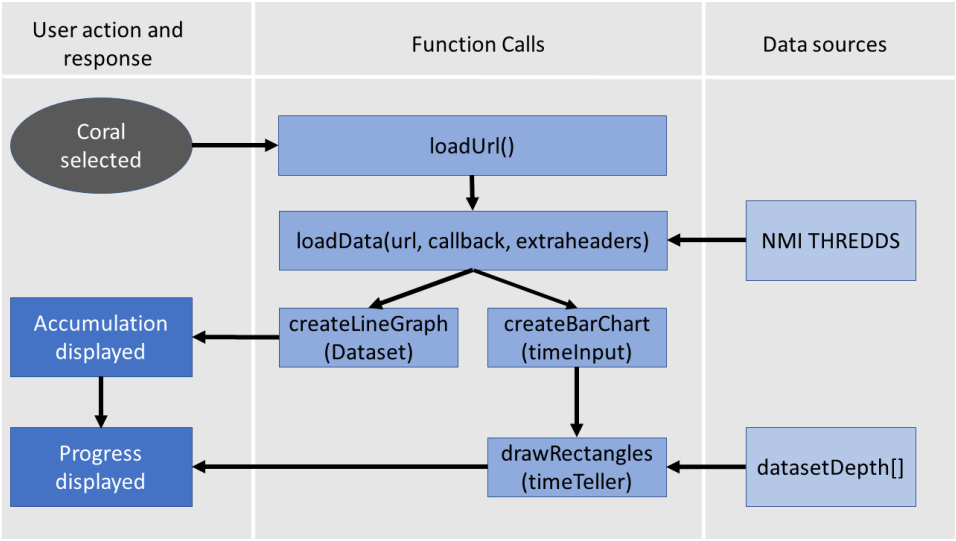

Figure 6.9: Coral selected user action and response

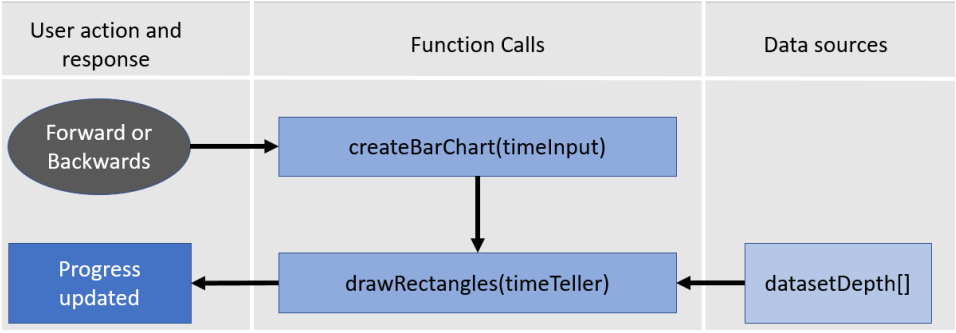

Figure 6.10: Forwards and backwards user action and response

6.2.4 Implementation

The complete implementation can be found at: `https://www.dropbox.com/sh/5r3m2nij5qxgqsn/AACE8m8rjTsjS7eflEujeULoa?dl=0`. To run the prototype all files must be downloaded into the same folder, and the Allow-Control-Allow-Origin Chrome extension must be downloaded and activated. The prototype depends on data from Norwegian Meteorological Institute's THREDDS server, so try to access the following link prior to running the prototype: `http://thredds.met.no/thredds/catalog.html`. A skeleton of loadData() was provided by Tor Nordam while jsdap.js, parser.js, vbscript.js and xdr.js. are taken from `https://github.com/omarbenhamid/jsdap`.

The codebase consists of three types of files; an HTML file describing the structure of the elements in the front-end prototype, a CSS file in charge of formatting the different elements in the HTML file and several JavaScript files containing functions that add dynamic functionality to the HTML elements. In this subsection the main structure of the front-end will be examined, before the implementation of loadUrl(), loadData() and drawRectangles() are discussed.

Main Structure

The screen is divided into three main areas; the top container, left body and right body. Figure 6.12 shows the HTML implementation of the prototype. As seen, both the left body and the right body are further divided into elements described by the ”class” attribute and identified by the ”id” attribute. The class attribute defines the styles (CSS) applying to that element, while the id attribute makes it possible for Javascript to access and manipulate the elements.

The left body contains the wellbore drop down menu, a place holder for the image representing the wellsite, and the drop down menu containing the coral elements. The right body contains two placeholders, the fist for the accumulation chart and the second for the progress chart, in addition to the ”backwards” and ”forward” buttons.

As the user completes different user actions the placeholders are filled in with the appropriate elements and information. Figure 6.11 show an example of how the placeholder for the accumulation chart looks before and after the user has selected a coral. When a coral is selected all the elements comprising the accumulation chart is added dynamically to the HTML document. The placeholders used for the accumulation chart and progress chart is Scalable Vector Graphic (SVG) elements. The main reason to use SVGs as placeholders, as opposed ”divs”, is that they scale their content without loosing the integrity and visual consistency of the elements that comprise the SVG.

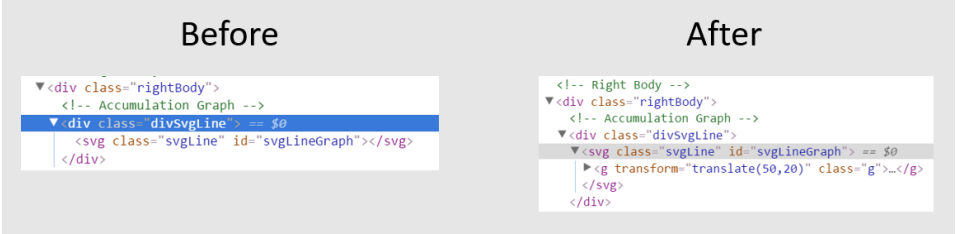

Figure 6.11: Placeholder before and after coral is selected

loadUrl() and loadData()

Simply put, loadUrl() and loadData() are in charge of obtaining data from the Norwegian Meteorological Institute's THREDDS server, parse the response, and pass it on to createLineGraph. The implementation of loadUrl() and loadData() can be viewed in Figure 6.15, and a stepwise explanation follows below.

One, the correct URL is collected from coralValueArray[], given the index of the selected coral in the coral drop down menu. Figure 6.14 show an example of a URL. The URL uses the following format: [start:step:stop] to define the time interval between each data point (green), and which grid cells to collect information from (blue). The URL shown in Figure 6.14 specifies that the returning JSON file should contain one data point from the period between 00.00 to 23.00 at depths between 0 and 0 (top layer), at grid cell id x=800 and y=10. The section underlined with red indicates the collection date for the data. Because data sets are only stored at the THREDDS server for a certain amount of time (approximately 10 day), the dates must be manually updated in coralValueArray[] when they expire.

Two, the URL is used as an input parameter in loadData(), together with inputData. loadData() connects to the THREDDS server and finds the specified data and stores it in inputData, which is a callback parameter. An example of a JSON response stored in inputData can be viewed in Figure 6.13, where the velocities are underlined with blue and the time is underlined with red.

Three, extract the necessary data from the JSON response. Because each velocity data point is enclosed by four brackets two steps of extraction was required to get each velocity data as a separate value in the velocity[] array. The time data could be transferred straight from the JSON response to the time[] array.

Four, format data. Because the procedures implemented in createLingegraph() aren't compatible with strings, the values of time[] and velocity[] are parsed into integers. Then each associated time-velocity pair are pushed into dataset[], before the values representing time are transformed from Unix Time to normal time and dates. None of the last two steps are necessary, but simplifies some of the procedures in crateLinegraph() and makes the data easier to relate to for the user.

```
68 <!-- Top Container -->
69 <div class="topCon">
70     <button class="menyButton"> Menu </button>
71 </div>
72
73 <!-- Left body -->
74 <div class="leftBody">
75
76     <!--Drop down menu: wellbore -->
77     <div class="divArea">
78         <form class="formCoral" id="wellboreForm" method="post" action="#">
79             <select class="selectArea" id="areaDropdown">
80                 <option value="0" selected="selected">Wellbore</option>
81             </select>
82         </form>
83     </div>
84
85     <!--Placeholder for the image -->
86     <div class="divMap" id="divImg">
87     </div>
88
89     <!--Drop down menu: coral -->
90     <div class="divCoral">
91         <form class="formCoral" id="coralForm" method="post" action="#">
92             <select class="selectCoral" id="coralDropdown" onchange=loadUrl()>
93                 <option value="0" selected="selected">Coral</option>
94             </select>
95         </form>
96     </div>
97 </div>
98
99 <!-- Right Body -->
100 <div class="rightBody">
101
102     <!-- Accumulation Graph -->
103     <div class="divSvgLine">
104         <svg class="svgLine" id="svgLineGraph"></svg>
105     </div>
106
107     <!-- Progress Chart -->
108     <div class="divSvgProg">
109         <svg class="svgProgress" id="progressChart"></svg>
110
111         <!-- Forward and Backwards buttons -->
112         <form method="post" action="#">
113             <button type="button" onclick=createBarChart(-1)> Back </button>
114             <button type="button" onclick=createBarChart(1)> Forward </button>
115         </form>
116     </div>
117 </div>
118
```

Figure 6.12: HTML implementation of the prototype

{"type":"Dataset","attributes":{},"u":{"type":"Grid","attributes":{},"array":
{"type":"Int16","attributes":{},"name":"u","dimensions":["time","depth","Y","X"],"shape":
[24,1,1,1],"id":"u.u"},"maps":{"time":{"type":"Float64","attributes":{},"name":"time","dimensions":
["time"],"shape":[24]},"depth":{"type":"Float64","attributes":{},"name":"depth","dimensions":
["depth"],"shape":[1]},"Y":{"type":"Float64","attributes":{},"name":"Y","dimensions":["Y"],"shape":
[1]},"X":{"type":"Float64","attributes":{},"name":"X","dimensions":["X"],"shape":
[1]}},"name":"u","id":"u","data":[[[[[-314]]],[[[-268]]],[[[-203]]],[[[-74]]],[[[74]]],[[[182]]],
[[[237]]],[[[216]]],[[[139]]],[[[7]]],[[[-143]]],[[[-283]]],[[[-376]]],[[[-383]]],[[[-302]]],
[[[-154]]],[[[20]]],[[[186]]],[[[299]]],[[[318]]],[[[258]]],[[[137]]],[[[-17]]],[[[-172]]]],
[1485648000,1485651600,1485655200,1485658800,1485662400,1485666000,1485669600,1485673200,1485676800,
1485680400,1485684000,1485687600,1485691200,1485694800,1485698400,1485702000,1485705600,1485709200,1
485712800,1485716400,1485720000,1485723600,1485727200,1485730800],[0],[-1144000],
[-3184800]]},"name":"fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2017012900.nc","id":"fou-
hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2017012900.nc"}

Figure 6.13: The THREDDS Response

http://thredds.met.no/thredds/dodsC/fou-hi/norkyst800m-1h/NorKyst-800m_ZDEPTHS_his.an.2017010700.nc.dods?u[0:1:23][0:1:0][800:1:800][10:1:10]

Figure 6.14: Link used to access NMI THREDDS server

drawRectangles()

drawRectangles() renders the progress chart by using functionality provided by D3. The complete implementation of drawRectangles() can be viewed in figure 6.16. A proper result is achieved by going through a series of steps which are basically equal for rendering any kind of chart.

One, determining height, width and margins. The height and width are used to set the height and width of the SVG element. The margins indicates how far from the edges the different components comprising the bar chart should be drawn. Two, create an empty SVG element based on the height and width determined in the previous step. To make sure the SVG element appears at the desired location this is specified by selecting the location to append the SVG element.

Three, create the scale functions in X and Y direction. Because data sets are unlikely to correspond exactly to pixel measurements in the visualisation, scale functions are used to map from an input domain to an output range (Murray 2015). To be able to do the mapping the input domain and output range must be specified. The input domain in X direction is specified setting the minimum value to zero and the maximum value equal to the maximum value in datasetDepth[].

The input domain in Y direction is specified to be the name (datatype) of the different JSON objects in datasetDepth[]. By using the scaleBand(), rather than scaleLinear(), in the Y-diection the input domain is transformed into discrete uniform bands in the output range. The padding parameter indicates the distance between each band. The range in both X and Y direction are specified using the height, width and margins.

Four, define axes. Axes are defined as either left, right, top or bottom based on where the text is displayed relative to the line representing the axes. In this solution a left and bottom axes are used. The axes are then scaled by their respective scale functions. Five, render axes. The axes are added to the SVG element by appending an element ”g” containing the axes. A class is added to the g element enabling styling using CSS. The transform function is used to arrange the axes in the correct position, as default position is origin.

```
24 function loadUrl() {
25     // Gets URL from the coralValueArray
26     var url = document.getElementById("coralDropdown")
27     .options[document.getElementById("coralDropdown")
28     .selectedIndex].value;
29
30     //Loads the JSON response associated with the URL
31     //to the inputData parameter
32     loadData(url, function(inputData) {
33         var dataset = [];
34         var velocityTemp = [];
35         var velocity = [];
36
37         //Gets the length of the array
38         //containing the velocity values
39         var velLength = inputData.u.data[0].length;
40
41         //Populate the time array with the
42         //times from the JSON response
43         var time = inputData.u.data[1];
44
45         //Populate the velocity array with
46         //the velocities from the JSON response
47         for(i=0; i < velLength; i++){
48             velocityTemp[i] = inputData.u.data[0][i][0][0];
49             velocity[i] = velocityTemp[i][0];
50         }
51
52         //Translate from JSON strings to integers
53         velocity = velocity.map(function(d){return Number(d);});
54         time = time.map(function(d){return Number(d);});
55
56         //Add associated time and velocity pairs
57         //to the dataset array
58         for (var i = 0; i < velocity.length; i++){
59             dataset.push({x:time[i], y:velocity[i]});
60         }
61
62         //Convert from UNIX timestamp to "normal time"
63         dataset.forEach(function(d){d.x = new Date(d.x*1000);});
64
65         createLinegraph(dataset);
66         createBarChart(0);
67
68         });
69     return false;
70 }
```

Figure 6.15: Implementation of loadUrl() and loadData()

Six, draw the rectangles representing the bars in the progress chart. The first four lines specifies that rectangles should be drawn for each entry in datsasetDepth[]. Because datasetDepth[] consists of the two JOSN objects, two rectangles will be rendered. The four attributes indicates the location (start point) of the rectangles and their height and width. Because the bars in the progress chart are horizontal, width is used to determine the ”height” of the bars and vice versa. The timeTeller parameter indicates the entry value in the JSON object's arrays used to calculate the width of the bar.

6.3 Analysis

There are several deviations between the target solution and the prototype solution. The most noticeable are the lack of the information tables and risk threshold values in the accumulation chart. Other features in the target solution are substituted by simpler features in the prototype solution, containing the same functionality. For instance the interactive map feature in the target solution is replaced with an image and a drop down menu in the prototype.

Although some of the features, such as the tables, are easily added to the prototype as HTML elements, the functionality would require increased the level of dummy data. Dummy data already play a significant role in the prototype solution, and as the features in question are not considered to be crucial for the main functionality, adding more dummy data was not desired. Sometimes developing features from scratch are more favourable than adapting features that needs comprehensive modifications, which is believed to be the case in this situation.

During the development phase it was not possible to get access to ELMO and proper data, as a result development and testing of the prototype solution is based on dummy data. Because there is a deviation between the structure of the dummy data and the structure of the associated data in ELMO, significant modification of the prototype must be done to make it comply with formats and structures used in ELMO.

The prototype solution use basic arrays to store most of the dummy data, with the exception of datasetDepth[] which is a JSON array. To make the prototype resemble the target solution several improvements are required. As explained in chapter 4, the front-end solution receives the model results as JSON responses, therefore transforming the current basic arrays into JSON arrays and objects is the first step towards the target solution.

The second step is to study the Climate and Forecast Metadata Convention and CF-NetCDF. This will give insight into how data is stored in ELMO and how to access the correct fields and attributes. Studying the WITSML is also necessary to get insight into the standard used for drilling data, which is displayed in the progress chart. The JSON responses received from ELMO are assumed to correspond with the above mentioned standards, and should be reflected in the local JSON structures in the prototype.

To comply with these changes, the functions collecting data from the arrays needs to be modified to deal with JSON arrays and objects, and to produce the right input for the functions rendering the charts. It is assumed that some of the procedures and knowledge used in loadData() can be adopted and applied to modify the functions in question.

```
40 function drawRectangles(timeTeller){
41
42     // Set Width and height
43     var margin = {top: 20, right: 20, bottom: 20, left: 60};
44     var width = 500;
45     var height = 250;
46
47     // Creating an empty SVG
48     var svg1 = d3.select("#progressChart")
49     .append("svg")
50     .attr("width", width)
51     .attr("height", height);
52
53     //Get maximum depth
54     var allDepths = datasetDepth[0].depth.concat(datasetDepth[1].depth)
55     var xMax = d3.max(allDepths, function(d){return d;});
56
57     //Create range/scale functions
58     var xRange = d3.scaleLinear()
59     .range([margin.left, width - margin.right])
60     .domain([0, xMax]);
61
62     var yRange = d3.scaleBand()
63     .range([height - margin.top, margin.bottom])
64     .padding(0.2)
65     .domain(datasetDepth.map(function(d){return d.datatype;}));
66
67     //Define X-axis
68     var xAxis = d3.axisBottom()
69     .scale(xRange);
70
71     //Define Y-axis
72     var yAxis = d3.axisLeft()
73     .scale(yRange);
74
75     //Draw X-axis
76     svg1.append("g")
77     .attr("class", "axis")
78     .attr("transform", "translate(0," + (height  - margin.bottom)+ ")")
79     .call(xAxis);
80
81     //Draw Y-axis
82     svg1.append("g")
83     .attr("class", "axis")
84     .attr("transform", "translate("+(margin.left)+",0)")
85     .call(yAxis);
86
87     //Draw the rectangles
88     svg1.selectAll("rect")
89     .data(datasetDepth)
90     .enter()
91     .append("rect")
92     .attr("x", margin.left)
93     .attr("y", function(d){return yRange(d.datatype);})
94     .attr("width", function(d){return(xRange(d.depth[timeTeller]) -
   (margin.left));})
95     .attr("height", yRange.bandwidth());
96 }
```

Figure 6.16: Implmentation of drawRectangles()

6.4 Conclusion

The functionality presented in the target solution will be a useful decision support tool during drilling operations. The combination of the accumulation chart and progress chart gives a comprehensive picture of the state of the chosen corals, as well as the damage further drilling may cause. It is believed that is is necessary to present these charts together to achieve the desired level of decision support, and that displaying only the accumulation chart would not be adequate.

The deviation between the target solution and the prototype solution ended up being bigger than originally planned. The main reason is lack of access to ELMO and related data during the development and testing phase. It proved to be more difficult than anticipated to get a good understanding of all the required data sources and the relations between the data. The prototype is a starting point for developing a solution with requested functionality, but significant tailoring is required before deploying it into a production environment.

Chapter 7

Conclusion

Developing a framework for analysing integration architectures in systems-of-systems was more challenging than expected, due to the vast amounts of available literature. The main challenged evolved around correlating information from several sources, using different terminology, having different target areas and covering partly the same areas.

Creating a comprehensive framework ensured that it would cover ELMO's integration architecture, and that the framework could provide references and a wider perspective when evaluating ELMO. The framework gives good insight into challenges, options and complexity drivers that have to be evaluated when establishing an integration architecture for a system-of-systems.

The framework focus on System design, Scope and development context, Management of common and shared data, Control regimes for cross-system activities and System connections. Other important areas such as security and infrastructure technology has not been possible to cover in this thesis, but are just as important areas.

When the framework was completed, analysing ELMO was a straightforward process, indicating that the framework fulfilled its purpose. The initial impression was that ELMO had a rather advanced integration architecture, but the analysis proved that ELMO adopts a relatively basic integration architecture.

The integration architecture enables ELMO access functionality available in the individual systems and deliver new business functions. By applying what is believed to be a stable information model, through the use of Climate and Forecast Metadata Convention and NetCDF, ELMO is well prepared for potential expansions, such as multiple parallel operations.

The practical part of the thesis produced a functional front-end prototype. Although there is a gap between the target environment and the prototype solution, the main purpose of the task is considered to be achieved. The practical project has given good insight into how basic tools, such as HTML, CSS and JavaScript can be used to develop dynamic front-end solutions and the importance of having access to specific libraries for handling specific tasks, such as D3.

The main limitation for the completion of the front-end prototype is considered to be

the lack of access to important documentation, test environment and proper data. On the other hand, it is believed that the functionality presented in the target environment will be useful during drilling processes as decision support.

Bibliography

Ahsan, K. & Nurmilaakso, J.-M. (2015), 'A novel three-layer architecture for information system integration'. Accessed January 13th 2017.[Online]. Available: https://www.researchgate.net/profile/Juha_Miikka_Nurmilaakso/publication/280035431_A_novel_three-layer_architecture_for_information_system_integration/links/55e40b3b08ae2fac4721416d.pdf.

Bass, L., Clements, P. & Kazman, R. (2013), *Software Architecture in Practice*, third edition edn, Addison Wesley.

Bostock, M. (2017), 'Data-driven documents'. Accessed January 7th 2017.[Online]. Available: https://d3js.org/.

Brechan, B., Hovda, S. & Skalle, P. (2016), 'Compendium. introduction to drilling engineering', Trondheim. Department of Petroleum and Applied Geophysics, NTNU.

Brönner, U., Nepstad, R., Eidnes, G., Rønningen, P. & Rye, H. (2013), 'A real-time discharge modelling and environmental system for drilling operations'. SPE European HSE Conference and Exhibition.

Brönner, U., Nordam, T., Alver, M. O., Eidnes, G., & Aarnes, Ø. (2015), Real-time monitoring and modelling of drilling operations in sensitive areas. elmo project - results. Internal technical report.

Brönner, U., Nordam, T., Alver, M. O., Michelsen, F. A., Møskeland, T. & Aarnes, Ø. (2016), Real-time decision support by combination of monitoring and modelling through open standards. Submitted to Elsevier.

DNV GL (2014), 'Drilling in sensitive areas'. Accessed December 22st 2016.[Online]. Available: https://issuu.com/dnvgl/docs/drilling_in_sensitive_areas_dnvgl.

El-Sappagh, S. H. A., Hendawi, A. M. A. & Bastawissy, A. H. E. (2011), 'A proposed model for data warehouse elt processes', *Journal of King Saud University - Computer and Information Science* **23**(2), 91–104.

Foss, J. H., Mortensen, P. B. & Furevik, D. M. (2002), 'The deep-water coral lophelia pertusa in norwegian waters: distribution and fishery impacts', *Hydrobiologia* **471**(1), 1–12.

Fowler, M. (2002), *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional.

Hepsø, V., Låte, M., Gramvik, G., Johnsen, S., Nilssen, I., Wesenberg, H. & Statoil ASA (2012), 'Integrated environmental monitoring in daily operations'. SPE Intelligent Energy International.

json.org (2017), 'Introducing json'. Accessed January 7th 2017.[Online]. Available: `http://www.json.org/`.

Kazman, R., Nielsen, C. & Schmid, K. (2013), Understanding patterns for system-of-systems integration, Technical report, Software Engineering Institute, Carnegie Mellon University.

Larsson, A. I. & Purser, A. (2011), 'Sedimentation on the cold-water coral lophelia pertusa: Cleaning efficiency from natural sediments and drill cuttings', *Marine Pollution Bulletin* **62**(6), 1159–1168.

Microsoft Corporation (2004), *Integration Patterns*, Microsoft Press.

Murer, S., Bonati, B. & Furrer, F. J. (2011), *Managed Evolution - A Strategy for Very Large Information Systems*, Springer-Verlag Berlin.

Murray, S. (2015), 'Scales'. Accessed January 27th 2017.[Online]. Available: `http://alignedleft.com/tutorials/d3/scales`.

Nordam, T. (2017), 'Personal notes'. Information provided during supervisor sessions and by mail.

Norwegian Petroleum Depeartmen (2016), 'Factpages, exploration and development wellbores - ordered by the year drilling was entered'. Accessed December 21st 2016.[Online]. Available: `http://factpages.npd.no/factpages/Default.aspx?culture=nb-no&nav1=wellbore&nav2=Statistics%7cEntryYear`.

Petrowiki.org (2015*a*), 'Drilling fluids'. Accessed December 22st 2016.[Online]. Available: `http://petrowiki.org/Drilling_fluids`.

Petrowiki.org (2015*b*), 'Drilling fluids'. Accessed December 22st 2016.[Online]. Available: `http://www.glossary.oilfield.slb.com/Terms/w/well_plan.aspx`.

Rye, H., Ditlevsen, M. K., Moe, J. A. & Løkken, M. (2013), 'A real-time discharge modelling and environmental system for drilling operations'. SPE European HSE Conference and Exhibition.

Sintef (2017*a*), 'Elmo project'. Accessed January 10th 2017.[Online]. Available: `https://www.sintef.no/projectweb/elmo/elmo-application/`.

Sintef (2017*b*), 'Elmo project'. Accessed January 10th 2017.[Online]. Available: https://www.sintef.no/projectweb/elmo/.

Sintef (2017*c*), 'Sinmod'. Accessed January 18th 2017.[Online]. Available: https://www.sinmod.no.

Slagstad, D. & McClimans, T. A. (2005), 'Modeling the ecosystem dynamics of the barents sea including the marginal ice zone: I. physical and chemical oceanography', *Journal of Marine Systems* **58**.

Sommerville, I. (2011), *Software engineering*, ninth edition edn, Pearson.

Stephens, R. (2015), *Beginning Software Engineering*, John Wiley and Sons.

Torgersen, T. (2016), Real-time environmental monitoring and modelling in the petroleum industry - elmo solution overview ans analysis. https://www.dropbox.com/sh/5r3m2nij5qxgqsn/AACE8m8rjTsjS7eflEujeULoa?dl=0.

Ulfsnes, A., Møskeland, T. & Aarnes, Ø. (2012), 'Coral risk assessment - tool development'. SPE APPEA Internatinal Conference on Health, Safety and Environment in Oil and Gas Exploration and production.

Ulfsnes, A., Møskeland, T., Frost, Tone Karin, Hepsø, V., Gramvik, G., Ute, B. & Rylandsholm, P. (2014), 'Towards integrated environmental monitoring'. SPE International Conference on Health, Safety, and Environment held in Long Beach, California, UAS.

Vassiliadis, P. & Simitsis, A. (2009), *Extraction, Transformation, and Loading*, Springer US, pp. 1095–1101.

Zhao, Y., Brown, R., Kramer, T. R. & Xu, X. (2011), *Information Modeling for Interoperable Dimensional Metrology*, Springer-Verlag London.