NTNU
Innovation and Creativity

Software agents applied in oil production

Lise Engmo
Lene Hallen

Master of Science in Computer Science

Submission date:	May 2007
Supervisor:	Harald Rønneberg, IDI
Co-supervisor:	Jørn Ølmheim, Statoil
	Einar Landre, Statoil

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The students should investigate the applicability of agent technology in oil production. This should be done through a proof-of-concept demonstrating how software agents can solve problems which are difficult to handle using traditional approaches. The focus should be on challenges found within control and optimisation of production wells.

Assignment given: 16. January 2007
Supervisor: Harald Rønneberg, IDI

Abstract

The current increase in instrumentation of oil production facilities leads to a higher availability of real-time sensor data. This is an enabler for better control and optimisation of the production, but current computer systems do not necessarily provide enough operator support for handling and utilising this information. It is believed that agent technology may be of use in this scenario, and the goal of this Master's thesis is to implement a proof-of-concept which demonstrates the suitability of such solutions in this domain. The agent system is developed using the Prometheus methodology for the design and the JACK Intelligent Agents framework for the implementation. A regular Java program which simulates the running of a very simplified oil field is also developed. The simulator provides the environment in which the agent system is put to the test. The agents' objective is to maximise the oil production while keeping the system within its envelope. Their performance is compared with results obtained by a human operator working with the same goal in the same environment. The metrics we use are the number of critical situations which occur, their duration, and the total amount of oil produced. The results indicate that the agents react faster to changes in the environment and therefore manage to reduce the amount and duration of critical situations, as well as producing more oil. This suggests a possibility of introducing an agent system to deal with some aspects of the production system control. This may contribute to a reduction of the information load on the operator, giving him/her more time to concentrate on situations which the agents are not able (or not allowed) to handle on their own.

PREFACE

This Master's thesis has been written as a continuance of a project which we performed in cooperation with Statoil during the autumn semester of 2006. The goal of the project was to investigate ways in which intelligent software agents could be used to reduce the information flow from instrumented production facilities. The characteristics of agents make them a tempting approach to address challenges related to the increased availability of real-time data, and we suggested four scenarios in which agents seem especially well-suited. During our work with this Master's thesis we have brought our findings from the project into more practical use. First, a specific problem area was chosen in cooperation with domain experts from Statoil. We continued our work with the design of an agent solution meant to meet the stated challenges, and implemented this solution as a proof-of-concept. The agent system was tested in a simulated environment and its performance compared to the results obtained by a human operator working in the same environment.

We would like to thank Tor Gunnar Aksland at Statoil for helpful explanations of the oil production processes, our friends at Ugle for always sharing their coffee with us, and our supervisors Harald Rønneberg, Jørn Ølmheim and Einar Landre at Statoil for all help and support during our work with this thesis. Their enthusiasm for the topic and interest in our suggested solutions have been very motivating.

Trondheim, 30 May 2007

[illegible] Lene Hallen

Lise Engmo   Lene Hallen

CONTENTS

List of Tables

LIST OF FIGURES

Listings

CHAPTER 1

INTRODUCTION

The main objective with this chapter is to set the scene for the rest of the thesis. We begin in Section 1.1 by explaining the motivation behind the choice of topic. The challenges mentioned here are further explained in the problem definition of Section 1.2. This section also contains the project goal which represents the essence of what we aim for. The project context in Section 1.3 gives a very brief introduction to Statoil as a company and their interest in the results of this work. Section 1.4 contains an outline of the rest of the report.

1.1 Motivation

The agent paradigm is a fascinating, promising, and relatively new way of conceptualising, designing and implementing software systems. Our interest in this subject was invoked in the fourth grade course *Distributed Artificial Intelligence and Intelligent Agents*, and the project work we performed for Statoil during the autumn semester of 2006 increased our enthusiasm for the topic even more. While the introductory course provided the theoretical foundations for understanding the benefits and challenges of agent technology, the project work gave us the opportunity to investigate its possible practical use in the context of oil production systems.

The challenges addressed in our project work were all related to the current increase in instrumentation of Statoil's production assets. A higher degree of instrumentation is believed to reduce operating and maintenance costs, but our hypothesis was (and is) that higher availability of real-time sensor data might also lead to information overload for the human operators and current control systems. The goal of our project was to investigate ways in which multiagent systems could be used to alleviate potential problems related to information overload, and our project report [10] summarises our findings. For our Master's thesis we will continue the same line of thought, based on the problem definition stated in the next section.

1.2 Problem definition

The following problem definition has been formulated in cooperation with our teaching supervisors at Statoil:

> *The students should investigate the applicability of agent technology in oil production. This should be done through a proof-of-concept demonstrating how software agents can solve problems which are difficult to handle using traditional approaches. The focus should be on challenges found within control and optimisation of production wells.*

Based on this problem definition and further discussion with our teaching supervisors and domain experts at Statoil, the following goal was defined:

> *Design and implement a proof-of-concept which demonstrates how a multiagent system may contribute in the control and optimisation of production wells in order to reduce the information load on human operators.*

1.3 Project context

This Master's thesis has been written in collaboration with Statoil, a Norwegian-based oil and gas company with more than 25,000 employees in 33 countries worldwide. In a press release dated 15. December 2005, it was announced that Statoil *is developing software to support oil trading and operations management, using the JACK Intelligent Agent development toolkit from Agent Oriented Software (AOS)* [9]. Agent technology is still fairly new at Statoil and the IT Department is eager to learn more about how the use of agents may contribute positively in their systems.

We are expected to use the JACK Intelligent Agent development toolkit in our work with this thesis, and this has had implications for our choice of methodology and design tools.

1.4 Report outline

We start with short introductions to agent technology, our choice of methodology and tools, and the application area. Our agent system is developed using the Prometheus methodology and the design and implementation are documented in Chapter 7 and 8. The final part of our report contains our results and conclusions.

Chapter 2, Software agents Agent-oriented software engineering is a paradigm which provides a framework for effective communication and reasoning about complex software systems on the basis of mental attitudes. The approach seems especially suitable for building complex distributed systems. In this chapter, we give a short introduction to the main concepts involved.

Chapter 3, Methodology and tools The development of multiagent systems is different from the development of conventional software systems. For that reason, a methodology especially suited for this type of software engineering should be used, and we have chosen to base our work on the Prometheus approach. In this chapter we also present the tools we use for designing and implementing our agent system.

Chapter 4, Application area In order to design an agent solution, it is important to understand the domain and its challenges. The goal of this chapter is to give a very brief introduction to the main steps involved in oil production. It also includes a section which explains why we believe that agents are suitable in this particular scenario, and what assumptions underlay our approach.

Chapter 5, Experiment approach Our agent system must be tested in order for us to draw any conclusions on whether or not it works as intended. The most important content of this chapter includes the experiment process and definition, formulations of our hypotheses, and an evaluation of validity threats.

Chapter 6, Simulated environment We develop a test framework which simulates the running of a simplified oil field. In this chapter we describe the main features of this simulator, with emphasis on how we define environmental influence and generate the sensor values.

Chapter 7, System specification The objective in this initial phase of Prometheus is to identify the goals and basic functionalities of the agent system. We start with a high-level description of the system to be designed and present the goals of the system as a decomposition of the main goal. Scenarios are used as a tool to ensure that all functionality is covered. The chapter also includes the definition of *roles*; groupings of functionalities with related inputs and outputs.

Chapter 8, System development The artefacts produced during the system specification are used as a basis for the architectural design; a high-level design of the agent system. The overall system structure presented in this chapter is probably the most important and useful artefact resulting from the initial two phases of the Prometheus methodology. The chapter ends with a short description of the detailed design and some implementation remarks.

Chapter 9, Results Our agent system is tested in a simulated environment and its performance compared to the performance of a human operator in the same environment. The results from each of the testruns are presented in tables and graphs, and the chapter ends with a discussion leading to the rejection or acceptance of the stated hypotheses.

Chapter 10, Summary The background and motivation, our suggested solution and the results of our experiment are here briefly summarised.

Chapter 11, Conclusion Based in our original problem definition, we here consider the further implications of our results.

Chapter 12, Further work We here present some possible approaches for further work.

Part I

State of the art

CHAPTER 2

SOFTWARE AGENTS

We begin this chapter with Section 2.1 which lists some typical characteristics of complex software systems. The agent concept is then defined in Section 2.2 and Section 2.3 gives an introduction to the Belief-desire-intention (BDI) architecture, one of the most influential approaches to the study of agent-oriented systems. Multiagent systems consist of a number of software agents, each one capable of interacting with the others. The main characteristics of such systems are presented in Section 2.4. The chapter is concluded with a summary of some known benefits and challenges associated with the agent-oriented approach.

2.1 The nature of complex software systems

Complexity is an innate property of the types of task for which software is used. A wide range of software engineering paradigms have been devised in order to reduce this complexity to a manageable level, and agent-oriented software engineering is one such paradigm.

Complex systems exhibit a number of important regularities [12];

- Complexity often takes the form of a hierarchy, i.e. the system is composed of inter-related subsystems, each of which is itself a hierarchy. These organisational relationships are not static and their precise nature varies between subsystems.
- The choice of which components in the system are primitive is relatively arbitrary and is defined very much by the observer's aim and objectives.
- A complex system will evolve from a simple system more quickly if there are stable intermediate forms, than if there are not.
- It is possible to distinguish between the interactions among subsystems and the interactions within subsystems. Complex systems are seen as nearly decomposable; subsystems can be treated almost as if they are independent of one another, but not quite since interactions exist. Although many of these interactions can be predicated at design time, some cannot.

Software engineers have developed a number of powerful tools in order to manage complexity, the main mechanisms are *decomposition*, *abstraction*, and *organisation*. Agents represent a natural abstraction mechanism with which to decompose and organise complex systems. Adopting an agent-oriented approach means to decompose the given problem into multiple, interacting, autonomous agents that have particular objectives to achieve. The key abstraction models are agents, interactions, and organisations [12].

2.2 Agent characteristics

The following definition of the agent concept has been proposed by Wooldridge in [25]:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.

To better understand what an agent is and how it can be used, we need to identify its characteristics. The definition above states an agent's main features; it is situated in an environment and it is autonomous. These two properties alone do not imply intelligence. An intelligent agent has one or more goals that it attempts to achieve through performing actions. It has some sort of perception of its environment and takes these observations into account when it decides which action(s) to perform. An intelligent agent might also communicate with other agents if it needs help to achieve a goal or it recognises that another agent has conflicting goals. A more thorough explanation of agents, including comparisons with passive objects and expert systems, can be found in [10].

Table 2.1 gives a summary of the properties characterising intelligent agents [25].

Situatedness	Agents sense the environment and their actions affect it.
Autonomous	Agents have some sort of free will and are independent. They decide for themselves what to do and with whom to cooperate. Internal states and goals form the autonomy.
Proactive	Agents have a goal which they take actions to pursue.
Reactive	Because agents have perception, they can react to changes in the environment when they find it necessary.
Social	Agents communicate with other agents, this includes negotiation and collaboration.
Flexible	Agents have several ways to achieve a goal. If they try to achieve their goal in one way and fail, they will try another way.
Robust	Agents can recover from failure. This characteristic is closely related to flexibility.

Table 2.1: Characteristics of intelligent agents

In addition to these characteristics, learning and intelligence are often seen as intimately related to each other. It is usually agreed that a system capable of learning deserves to be called intelligent, and conversely, that a system being considered as intelligent is expected to be able to learn [18].

The ability to learn enables agents to improve their performance over time. When designing an agent system it is very difficult to foresee all possible situations the agents might encounter and how they should act correspondingly. This especially applies to multiagent systems where global behaviour in many cases emerges rather than being pre-defined [1].

2.3 The Belief-desire-intention architecture

The behaviour of an agent is defined by its architecture. The architecture describes the internals of the agent entity; its data structures, the operations that may be performed on these data structures and the control flow between them. There are two main types of agent architectures; reactive and deliberative. A third category is the hybrid architecture which attempts to achieve the best from the two worlds [25].

BDI is an architecture belonging in the deliberative category. A BDI agent is an agent with three mental attitudes, stemming from theory on human practical reasoning. These mental attitudes determine the system's behaviour and are critical for achieving the required performance when deliberation is subject to resource bounds [16].

Beliefs Facts and assumptions about the world that the agent *believes* to be true. The beliefs may be incomplete or incorrect.

Desires The agent's goals; situations or objectives that it wishes to accomplish. They are what motivate it to act.

Intentions Represent the desires that the agent has committed to achieving, i.e. its deliberative state. Typically, the agent will continue to try to achieve an intention until it either believes that it is satisfied or believes that the intention is no longer achievable.

The agent has a symbolic representation of the world that is loosely coupled to these mental attitudes [25]. This is illustrated in Figure 2.1. The agent can maintain and manipulate this representation. In addition to the three mental attitudes, the agent has a structure which contains a set of plans. The plans specify the courses of actions which may be followed in order to achieve the agent's intentions. An *interpreter* is responsible for updating beliefs from observations made of the world, generating new desires on basis of new beliefs, and selecting intentions from the set of currently active desires. It also selects the actions to perform based on the agent's current intentions and procedural knowledge [5].

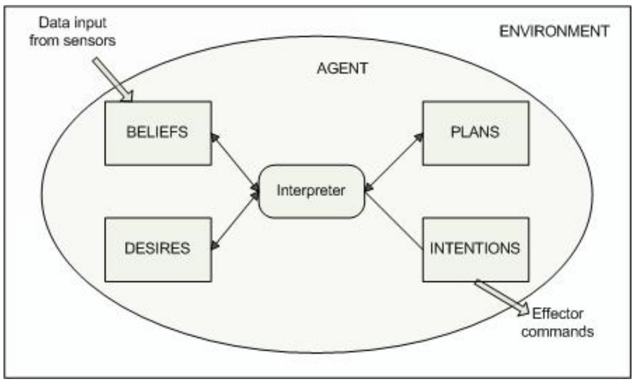


Figure 2.1: A BDI agent architecture

2.4 Multiagent systems

As explained in Section 2.2, an intelligent agent is an entity with its own set of goals and an ability to figure out for itself what it needs to do in order to achieve them. We have also stated that agents have the capability of social behaviour, i.e. agents can communicate with other agents.

A multiagent system consists of a number of such agents, each one capable of interacting with the others. In general, these agents act on behalf of owners with very different goals and motivations. They therefore require the ability to cooperate, coordinate, and negotiate with each other [25].

According to [20], the characteristics of multiagent systems are that:

1. each agent has incomplete information or capabilities for solving the problem,
2. there is no global system control,

3. data is decentralised,
4. computation is asynchronous

2.5 Benefits

The human-like characteristics of agents provide a high abstraction level which may simplify the modelling and implementation of systems for complex domains. Agents can be trusted to pursue their goals and take initiative to interact only when needed; this independence reduces the need for communication. Their autonomy leads to encapsulation of functionality, and coupling is reduced because agents do not provide any control point to external entities. The following list highlights some of the main dimensions along which agent systems are believed to enhance performance, these aspects are further elaborated on in [20]:

Computational efficiency because concurrency of computation is exploited. This requires that the communication is kept minimal, e.g. by transmitting high-level information and results rather than low-level data.

Reliability Components that fail can be gracefully recovered. Agents with redundant capabilities or appropriate inter-agent coordination are found dynamically and can take up responsibilities of agents that fail.

Maintainability A system composed of multiple components is easier to maintain because of its modularity.

Responsiveness The modularity of a multiagent system leads to the possibility of handling anomalies locally without propagating them to the whole system.

Flexibility Agents with different abilities can adaptively organise to solve a given problem. An agent can also have a number of plans for reaching its goal and adapt its strategy to changes in the environment.

In [13], the authors argue that *for certain classes of problem, adopting a multiagent approach to system development affords software engineers a number of significant advantages over contemporary methods*. If a problem domain is particularly complex, large or unpredictable, it might be that the only way it can be reasonably addressed is to develop a set of modular components that are specialised at solving a particular aspect of it. The characteristics of multiagent systems make them especially suited for meeting the challenges of such domains.

2.6 Challenges

The situatedness of agents makes it difficult to design software which is capable of maintaining a balance between proactive and reactive behaviour. Striking the balance implies context-sensitive decision-making which in turn means that there can be a significant degree of unpredictability in the system. Agents are autonomous, which means that the patterns and effects of their interactions are uncertain. Unpredictability in agent systems also relates to the notion of emergent behaviour; with a collection of processes acting side-by-side and interacting, behaviour which cannot be generally understood solely in terms of the individual components may emerge.

The notion of autonomous software components is not very comfortable for most users, and an agent system must be designed with this in mind. The balance must be struck between needlessly distracting the user and the agent exceeding its authority. It is also important to remember that agents may make globally sub-optimal decisions since complete global knowledge is not a possibility in most realistic agent systems.

CHAPTER 3

METHODOLOGY AND TOOLS

The design of multiagent systems differs from the design of conventional software systems, and the methodology used should reflect these differences. We have chosen to use the Prometheus methodology for the design of our agent system, and Section 3.1 gives an overview of its three phases. Prometheus Design Tool (PDT) is a tool which is especially designed for supporting software development following the Prometheus methodology. We will use this tool in the design of our system, and Section 3.2 presents its main features. JACK Intelligent Agents is an environment for building, running and integrating multiagent systems. JACK is presented in Section 3.3 where we also give a an introduction to the main agent-oriented concepts supported by this system. We use the Eclipse Software Development Kit (SDK) for the implementation of our system, and this development environment is presented in section 3.4.

3.1 Prometheus methodology

The Prometheus methodology has been developed with the goal of defining a process with associated deliverables for developing intelligent agent systems. Prometheus has been used internally at the company behind JACK and in numerous industry workshops and university courses. The methodology consists of three phases; system specification, architectural design, and detailed design. An overview of the phases are shown in Figure 3.1 [11]. It is important to realise that the methodology is intended to be interpreted as a set of guidelines, and that it must not necessarily be followed strictly. Iterative development is also a keyword here. The remainder of this section is based on [15] and is devoted to an introduction to each of the three Prometheus phases.

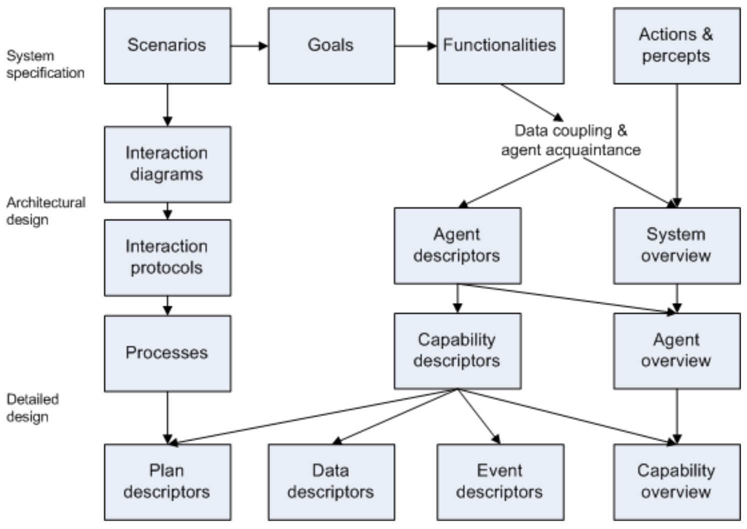


Figure 3.1: The phases of the Prometheus methodology

3.1.1 System specification

The focus in this phase is to identify system goals, develop use case scenarios illustrating the system's operation, identify the basic functionalities and specify the interface between the system and its environment in terms of actions and percepts.

System goals Goals are important because they state the reasons for building the system and because they are central to the functioning of the intelligent software agents that are going to realise the system. The starting point for building the initial list of system goals is the implicit indications provided in the system description. The initial goals are then refined into subgoals, and rearranged so that similar goals are grouped together.

Functionalities or roles Functionalities are chunks of behaviour, including a set of related goals, percepts, actions and data relevant to the behaviour. A functionality should be described adequately in one or two sentences. In addition to the natural language description and information about the goals and actions that are included, the functionality descriptor should also include *triggers*: information about events or situation that will initiate activity within this functionality. The Prometheus Design Tool uses the term *role* for functionality, and we will follow this convention.

Scenario development Scenarios show the sequences of steps that take place within the system and they are primarily used to illustrate the normal running of the system. As scenarios are developed, it becomes evident where there is a need for information from the environment and where actions are required. Additional goals are also often identified in this process.

Interface description How the agent system is going to interact with its environment is a question which has to be answered early. This implies identifying what environment input will be available to the agent system while it is running (percepts) and what the agent will do to interact with and affect the environment (actions). For percepts, it is important to take into account how the data is obtained, as well as the exact nature of the data and to what extent it can be processed to provide information of interest. Actions are also complex, often including monitoring for failure or continual feedback loops.

3.1.2 Architectural design

The artefacts resulting from the system specification phase are used as a basis for developing the high-level design of the agent system. This phase results in a definition of what agents are to be part of the system and how these agents interact in order to meet the required functionality of the system. The three steps followed are outlined below;

Agent types Defining the agent types involved in the system is done by considering the functionalities and scenarios, developing possible groupings of functionalities into agents, and evaluating these agents according to the criteria of coupling and cohesion. Each agent should be cohesive, and the agent system should be as loosely coupled as possible. Reasons for deciding to group certain functionalities could be that the functionalities seem related and that they require a lot of the same information.

Interactions The interaction between agents capture the dynamic aspects of the system. The specification of this interaction is done through three steps; developing interaction diagrams from use case scenarios, generalising these diagrams to interaction protocols, and developing protocol and message descriptors. The protocols define all interaction sequences which are valid within the system.

System structure First, the boundaries of the agent system and the interactions with other subsystems are defined. Then, the percepts and actions are described, as well as the relationships between these and the relevant agents. All shared data must also be defined, both external persistent data and internal shared data within the system. Finally, a system overview diagram is developed and checked for consistency.

3.1.3 Detailed design

The main goal of this phase is to flesh out the capabilities needed for each individual agent in order for it to fulfil its responsibilities as outlined in the functionalities it contains. Process specifications may also be developed in order to indicate the internal processing taking place within each agent. The refinement of agents in terms of capabilities gives the agent overview diagram and capability descriptors. These are analogous to the system overview diagram and the agent descriptors, but now the focus is on a single agent. In the same way, process specifications here provide a detailed view of an individual agent's part in a particular process. In the final stages of this phase, the capability descriptors are further developed to specify the individual plans, beliefs and events needed.

3.2 Prometheus Design Tool

The Prometheus Design Tool has been developed to support design and development of multiagent systems using the Prometheus methodology. Similar to most modern software engineering methodologies, Prometheus is intended to be applied in an iterative manner. This leads to a need for consistency checking in order to ensure that the design remains consistent when changes are made. Manual consistency checking is tedious and error prone, and tool support is therefore highly desirable [21].

The user interface of the PDT is shown in Figure 3.2. The program is written in Java and its main features include structured textual descriptors, information propagation from one part of the design to another, consistency checking, hierarchical views and report generation. The output diagrams of the detailed design phase can be readily transformed into JACK agent code.

Some aspects of the Prometheus methodology are still not supported by this tool. Future development work includes better support for protocol specification, support for process specification within agents, and integration of separate debugging tools [21].

3.3 JACK Intelligent Agents

JACK Intelligent Agents is a framework for building, running and integrating multiagent systems, which is based on the BDI model. It is a commercial product developed by The Agent Oriented Software Group. The development team consist of agent experts who have worked on two previous generations of agents systems, Procedural Reasoning System (PRS) and The distributed Multi-Agent Reasoning System (dMARS). PRS is a well known agent architecture rooted in the BDI model and dMARS is an implementation of that architecture [8]. The JACK Development Environment (JDE) is implemented in Java and offers its own language, the JACK Agent Language, which extends Java with agent-oriented concepts such as agents, plans, events and capabilities. The JDE offers three graphic tools [9]; the design tool, the plan editing tool and the plan tracer. The JACK compiler compiles the JACK code into regular Java source code before execution [8].

The Prometheus design concepts of agents, capabilities and plans map directly to JACK concepts, while internal and external messages in Prometheus map to different types of events in JACK. We here give a brief introduction to the entities available in JACK. The content of this section is based

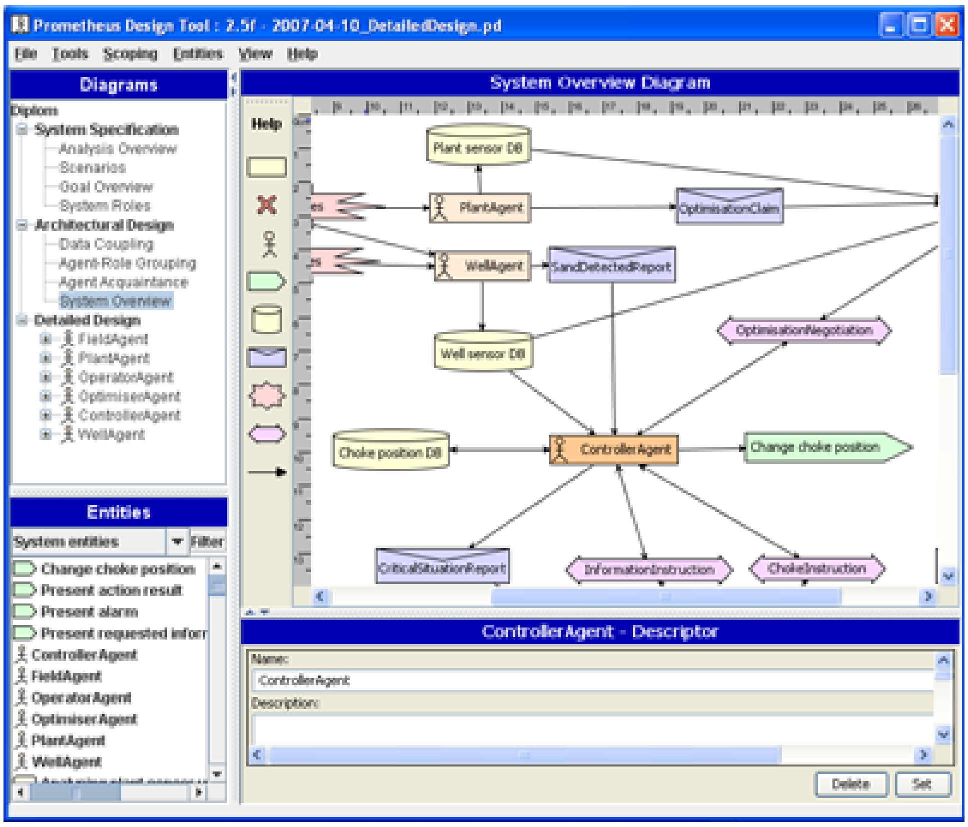


Figure 3.2: The Prometheus Design Tool user interface

on the JACK Agent Manual [6] and Agent Practicals [7], and more detailed explanations can be found there.

3.3.1 Agent

A JACK agent is a software component that can exhibit reasoning behaviour under both proactive and reactive stimuli. The following list shows the main content of a JACK agent together with the relevant code templates;

- Beliefs about the world
  `#(private|agent|global) data <Beliefset> beliefset_instance()`
- Capabilities that are groupings of plans
  `#has capability <Capability> capability_instance`
- Plans used to handle events
  `#uses plan <Plan>`
- Events that it will respond to
  `#handles event <Event>`
- Events that it may post to itself
  `#posts event <Event> event_instance`
- Events that it may send to other agents
  `#sends event <Event> event_instance`

3.3.2 Beliefset

Beliefset are used to maintain an agents beliefs about the world. A `ClosedWorld` beliefset only contains beliefs that the agent assumes to be true. All other statements about the world are assumed by the agent to be false. In the case of an `OpenWorld` beliefset, the set can contain beliefs that the agent assumes to be true and beliefs that the agent assumes to be false. All other statements are then defined by the agent as unknown. The list below contains the main elements of a JACK beliefset together with their respective code templates;

- Key fields that identify the beliefs in the set
  `#key field <FieldType> field_name`
- Value fields that represent information about this object that the agent needs to know
  `#value field <FieldType> field_name`
- Events that it may post to the parent agent
  `#posts event <Event> event_instance`
- Declarations of queries that the agent can perform
  ```
  #indexed query queryName(parameters)
  #linear query queryName(parameters)
  #complex query queryName(parameters) {method body}
  #function query <ReturnType> queryName(parameters) {method body}
  ```

3.3.3 Capability

A capability is a well-defined collection of plans, using particular beliefs or data, which address a specific set of goals for the agent. The following list shows the main content of a JACK capability;

- Internal events that it will respond to
  `#handles event <Event>`
- External events (messages from other agents) that it will respond to
  `#handles external (event) <Event>`
- Events that it may post internally within the capability
  `#posts event <Event> event_instance`
- Events that it may post outside the capability
  `#posts external event <Event> event_instance`
- Beliefsets or other data it uses
  `#(private|agent|global|exports|imports) data <DataType> data_name(arguments)`
- Plans
  `#uses plan <Plan>`
- Inner capabilities
  `#has capability <Capability> capability_instance`

3.3.4 Plan

A plan is triggered by an event; either an internal event or a message from another agent. When an event occurs, a `context` method can be used to determine if the plan is applicable for the situation. If so, the `relevant` method is used to determine if the plan is relevant for that specific event. The `body` is the main method of the plan. The code written here defines the steps of the plan. Each step is evaluated to true or false depending on whether the agents succeeds to actuate the step or not. If a step fails (is evaluated to false), the entire plan fails and the subsequent steps are not actuated. The following list contains the main elements of a JACK plan.

- Events that it will respond to
  `#handles event <Event> event_instance`
- Events that it may post to itself
  `#posts event <Event> event_instance`
- Events that it may send to other agents
  `#sends event <Event> event_instance`
- Beliefsets or other data the plan uses
  `#(uses|reads|modifies) data <DataType> dataname`
- A reasoning method
  `#reasoning method methodName(parameters) {method body}`
- Post-processing and clean-up steps when the plan has succeeded
  `#reasoning method pass() {method body}`
- Post-processing and clean-up steps when the plan has failed
  `#reasoning method fail() {method body}`
- Code that determines if the plan is relevant for the event instance
  `static boolean relevant(<EventType> event_instance) {method body}`
- A logical condition to determine if the plan is applicable
  `context() {method body}`
- The actual steps the plan performs
  `body() {method body}`

3.3.5 Event

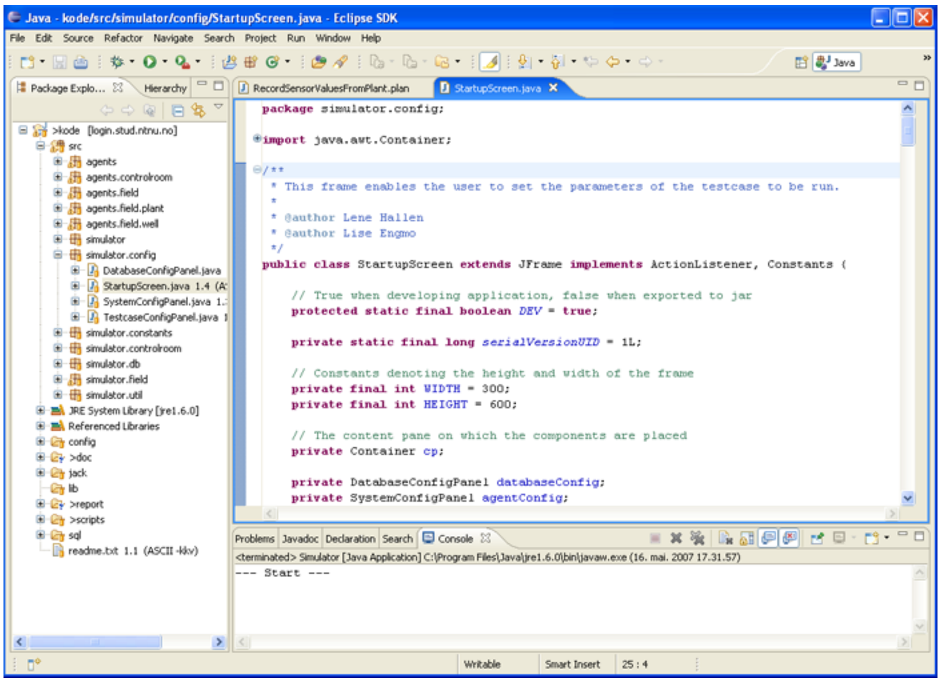
JACK defines several types of events and the ones most relevant for our work are briefly described here. `Event` is the base class for all normal events, and it can only be posted internally within an agent. A `MessageEvent` is an event that can be sent between agents. `BDIFactEvents` are internal events that allow for meta-level reasoning. They do not allow reconsideration of alternative plans if a plan fails. A `BDIMessageEvent` is equal to the `BDIFactEvent`, but can be sent between agents. `BDIGoalEvents` represent goals that an agent wishes to achieve. Meta-level reasoning and reconsideration of alternative plans if a plan fails is available. The following list contains the elements which are common for all JACK events.

- Beliefsets or other data the event uses
  `#uses data <DataType> dataname`
- Conditions (e.g. state of a belief) that should cause the event to be automatically posted
  `#posted when (condition)`
- How the event is posted
  `#posted as methodName(parameters) {method body}`

3.4 Eclipse

Eclipse is meant to be *a universal tool platform - an open extensible Integrated development environment (IDE) for anything and nothing in particular* [4]. Eclipse's Java development environment is what most people associate with the platform, and due to its plug-in architecture you may also use add-ons which enable development in several other languages. The standard version of Eclipse is composed of more than 80 plug-ins [2] and provides a number of useful features for programmers. A screenshot showing Eclipse's Java perspective is shown in Figure 3.3.

As mentioned in the previous section, JACK comes with a development environment especially intended for developing multiagent systems using the JACK agent language. The JDE provides good support for iteration between design and code, but we design our system in PDT and are therefore less dependent on this feature. JACKs development environment also lacks many of the features which we have grown accustomed to through using Eclipse in earlier projects, e.g. incremental code compilation, interfaces to standard source control systems, and code refactoring. The fact that we develop the simulator (which is a regular Java program) in parallel with the agent system, is another good reason for using Eclipse instead of JDE.


Java - kode/src/simulator/config/StartupScreen.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explo...
Hierarchy
RecordSensorValuesFromPlant.plan
StartupScreen.java
Problems
Javadoc
Declaration
Search
Console
Writable
Smart Insert
25 : 4

```
>kode [login.stud.ntnu.no]
  src
    agents
    agents.controlroom
    agents.field
    agents.field.plant
    agents.field.well
    simulator
    simulator.config
      DatabaseConfigPanel.java
      StartupScreen.java 1.4 (A
      SystemConfigPanel.java 1.
      TestcaseConfigPanel.java 1
    simulator.constants
    simulator.controlroom
    simulator.db
    simulator.field
    simulator.util
  JRE System Library [jre1.6.0]
  Referenced Libraries
  config
  >doc
  jack
  lib
  >report
  >scripts
  sql
  readme.txt 1.1 (ASCII -kkv)
```

```
package simulator.config;

import java.awt.Container;

/**
 * This frame enables the user to set the parameters of the testcase to be run.
 *
 * @author Lene Hallen
 * @author Lise Engmo
 */
public class StartupScreen extends JFrame implements ActionListener, Constants {

    // True when developing application, false when exported to jar
    protected static final boolean DEV = true;

    private static final long serialVersionUID = 1L;

    // Constants denoting the height and width of the frame
    private final int WIDTH = 300;
    private final int HEIGHT = 600;

    // The content pane on which the components are placed
    private Container cp;

    private DatabaseConfigPanel databaseConfig;
    private SystemConfigPanel agentConfig;
```

```
<terminated> Simulator [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe (16. mai. 2007 17.31.57)
--- Start ---
```

Figure 3.3: The Eclipse user interface

CHAPTER 4

APPLICATION AREA

The production of oil is an extremely complex undertaking, and the scope of this project does not allow us to delve deeply into the petroleum technical details. This chapter is therefore only meant as an overview of the most important steps in the processes involved in the production and processing of oil. We begin in Section 4.1 with a brief explanation of the main concepts involved in a typical oil production scenario. We continue in Section 4.2 with the motivation for choosing an agent-oriented approach to this specific application area. The chapter is concluded in Section 4.3 where we define our scope of work and the assumptions underlying our approach.

4.1 Oil production

Petroleum or crude oil is a liquid consisting of a complex mixture of hydrocarbons, and it occurs naturally beneath the earth's surface. This is illustrated in Figure 4.1. The petroleum can bubble to the surface in so-called "oil seeps", but can also be found several miles below the surface. Oil reservoirs typically extend over large distances, and full exploitation entails multiple wells scattered across the area [22]. Oil fields can be categorised by whether they are situated onshore or offshore. The principles are basically the same, but onshore wells are normally drilled vertically while offshore fields often have multiple directional wells drilled from a single well platform [3]. Some fields are extended with new fields to prolong the lifetime. These so-called satellite fields is an example of a characteristic which adds to the complexity of oil and gas developments. The well-fluid from all the fields must be gathered and transported to the processing facilities.

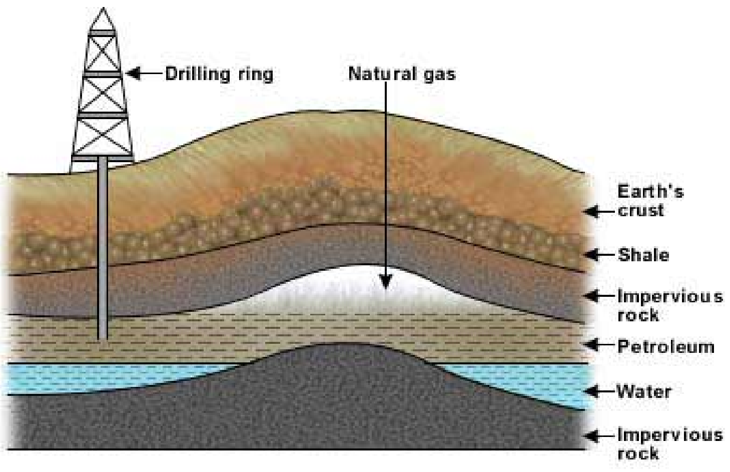


Figure 4.1: The occurrence of petroleum

4.1.1 Oil wells

Oil wells are perforations through the earth's surface designed to find and release petroleum hydrocarbons. A well is created by iteratively drilling a hole with an oil rig. For each iteration, a steel pipe slightly smaller than the hole is placed in the hole and secured with cement. This pipe is called a *casing*. With the casing in place, the formation so far is protected and one can continue drilling a new hole inside the first. Most modern wells have 2-5 sets of subsequently smaller holes drilled inside one another, each cemented with casing [3]. Figure 4.2 shows a sketch of a well with two casings.

When the hole is complete, the well must be perforated to let the oil in. Small holes are made in the casing that passes through the production zone [1]. How freely the oil flows into the well depends on the permeability of the surrounding formation. Often the well needs stimulation to provide the reservoir fluids with better access to the wellbore [3].

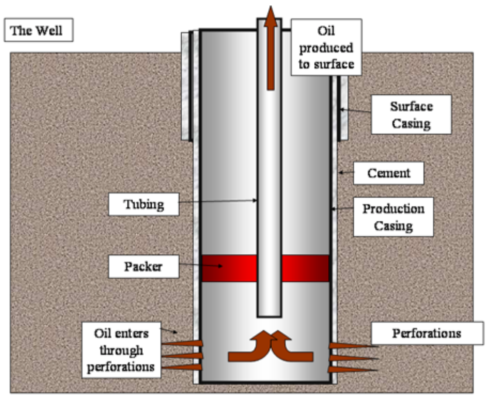


Figure 4.2: Sketch of a well, from [22]

A packer on the bottom of the tubing seals off between the tubing and the casing, and on top of the wellhead is a collection of valves that control the production flow. This valve manifold is called the Christmas tree and Figure 4.3 illustrates the valves it typically contains. The valves help regulate pressures, control flows, and allow access to the wellbore in case further completion work needs to be performed. The *choke* is an orifice that is used to control the well's flow rate. From the Christmas Tree, the flow can be connected to a distribution network of pipelines and tanks to supply the product to refineries, natural gas compressor stations, or oil export terminals.

4.1.2 Processing plants

The fluid which is pumped up from a well typically contains several waste products which must be removed before the crude oil can be further refined. The separation of oil from waste products is performed in production separators which are long horizontal cylinders, typically three metres in diameter and up to 25 metres in length, with a weir positioned close to one end. The well production enters as a mixture of oil, water, gas, and sand. As the fluid flow along the cylinder, gravity helps it separate into its constituent parts. The goal is to have within-specification oil flowing over the weir and uncontaminated water at the water outlet, but this is difficult to achieve because there are often no distinct interfaces between the various parts. Figure 4.4 illustrate the interfaces as they may appear between the various components of the well fluid [14]. After separation, the crude oil can be refined into more useful petroleum products. The refining processes are outside the scope of our work, and will therefore not be further explained.

[1]The *production zone* is the term used for the area in the earth stratum where the oil is located [3]

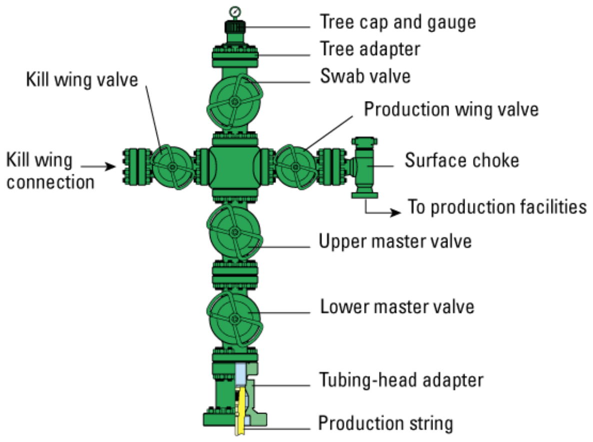


Figure 4.3: Christmas tree from [17]

Sand and water as unwanted constituents of a well's production are important elements of the environment in which our agent system will work. We therefore give a more thorough explanation of how they end up in the production and what problems they may cause.

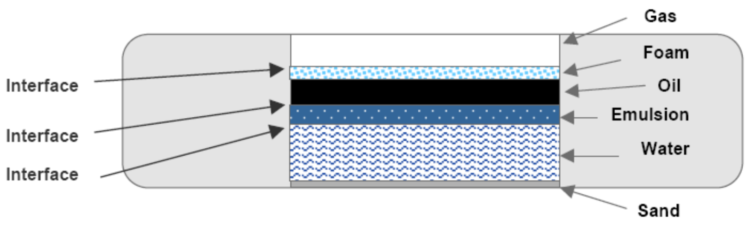


Figure 4.4: Profile based on different densities

Sand The amount of sand that enters the wellbore depends on the weakness of the rock in the production zone and how hard the well is produced. An example of a way to measure the sand content of the oil is mounting an acoustical gauge on the top of the well. The noise caused by the grains of sand hitting the gauge is measured and therefrom the total amount of sand can be estimated. The main problem with sand produced with high-velocity oil and gas streams is that it erodes any steel that it impinges on. Sand erosion is particularly severe where the flow stream changes direction. If an elbow cuts out, it can cause a blowout [2]. [3].

Water When oil is formed, it migrates into reservoir rocks and displace the water that occupies the porosity. This is a slow process and water will therefore always occupy some of the porosity, reducing the volume available for the oil. The oil saturation is vital for the possibility of extracting oil from the reservoir. If the oil saturation is too low, only water will flow into the well. For a well to produce oil only, the oil saturation must be at least about 80%. Consequently, many reservoirs produce both oil and water. The ratio between the two substances must be controlled during production, because the processing plant often has an upper limit on how much water it can handle [3].

[2]A blowout is when reservoir fluids blow up a hole and out onto the rig floor. The force of the fluids can smash tools together, causing a spark that could ignite the fluids and burn up everything in the vicinity [3]

4.1.3 Control and optimisation

In a typical oil production scenario, the processes and equipment of the wells and processing plant are monitored, controlled and optimised from a centrally located control room. In traditional systems, real-time data is difficult or impossible to obtain and the degree of automation is fairly low. Optimisation of the oil production is usually based on the use of mathematical models and is not performed very often, typically once a month. Future systems are likely to have a higher availability of real-time data, but operators and current computer systems are not necessarily able to take full advantage of this.

4.1.4 Example - Gullfaks

In order to give an impression of the size and complexity of a real oil field, we here present Gullfaks as an example. Gullfaks is one of Statoil's oil fields in the North Sea. It has three platforms (A, B and C) which receive crude oil from a total of eleven fields. In addition, platform A and B receive crude from three satellite fields [19]. The production is controlled for each platform and Table 4.1 shows the distribution of production wells and injection wells[3].

	Platform A	Platform B	Platform C
Main field production wells	27	30	35
Satellite field production wells	21		9
Injection wells	19	12	12

Table 4.1: Wells at Gullfaks

In total, there are 92 production wells at the main fields and 30 extra due to the satellite fields. All of these wells are mainly controlled manually. As the number of wells grows, this becomes an extremely difficult task. In shore-based heavy oil fields in Canada there may in fact be up to thousands of wells.

4.2 Motivation for an agent solution

For the control and optimisation of oil production, it is desired to automate more of the analysis and decision-making support. The operator should receive concrete advice rather than large amounts of raw data. In order to reduce the amount of data transmitted on communication links, we would like to distribute the analysis so that reasoning is performed closer to the data sources. Rather than sending a continuous stream of sensor data to a central repository, "intelligent" entities situated near the sensors should analyse the data and only send a notification when an anomaly occurs or when it is asked to do so.

Given the inaccessible location of oil wells, the distributed system must be robust and modular. It must be able to gracefully deal with module malfunction or breakdown, and software maintenance should be possible to perform remotely. The dynamic nature of the environment implies that the use of a static rule-based system is not adequate. It is impossible to create a set of rules which covers every possible situation that may occur. This calls for a system which is capable of flexible and autonomous behaviour. It should also have the ability to learn from what it experiences in order to adjust its behaviour accordingly.

There are several restrictions that must be met by the system. Local restrictions that differ between the wells must be dealt with individually, while at the same time ensuring that global restrictions

[3]An injection well is a well in which fluids, mainly gas or water, are injected rather than produced. The primary objective of such wells is typically to maintain reservoir pressure.

are met. In order to deal with the complexity of this situation, the responsibility for compliance with these restrictions should be distributed among different modules. Some kind of hierarchy must be implemented so that the most important restrictions are given higher priority than the others. The modules must be able to communicate to avoid conflicting actions being actuated.

In Chapter 2 we described intelligent software agents as being autonomous, proactive, reactive, social, flexible and robust, as well as situated in an environment which they can sense and act upon. These characteristics fit perfectly with the requirements we have just stated. Multiagent systems provide the modularity that we want and the agents' social ability makes them capable of meeting different restrictions and goals through negotiation and collaboration.

4.3 Scope of work

We will design and implement a proof-of-concept which shows that a multiagent system may be an advantageous approach to the challenges we have presented in this chapter. To be able to test our agent system, we need to simulate an oil production environment. Given the complexity of the domain and the scope of our work, it is necessary to place some limitations on this environment in which our agent system should operate. With the help of a domain expert, we have decided to concentrate on the challenges related to controlling the amount of sand and water in the production. We make the following assumptions with regards to the agents' environment:

- The production of the wells contains oil, water and sand, and no other substances.
- The wells will only have one valve each; a choke.
- The choke position of each well can be controlled by the operator from the control room.
- Permeability is an unpredictable environment variable and individual for each well's production zone.
- The total production rate of a well will only be influenced by the position of the choke and the permeability.
- Each well has a maximum limit of how much sand it can safely produce.
- The plant has a maximum amount of water that it can handle.
- Each well is equipped with sensors that measure sand, water and total production rate.
- The processing plant is equipped with sensors that measure water and total production rate.

Figure 4.5 illustrates our simplified environment containing three wells, a processing plant and a control room.

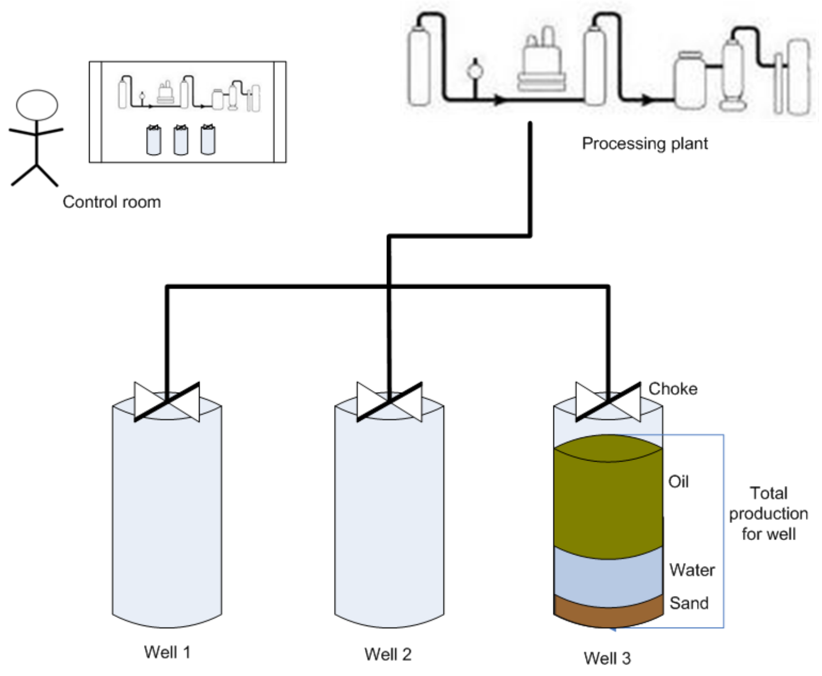


Figure 4.5: Our simulated oil field

Part II

Own contribution

CHAPTER 5

EXPERIMENT APPROACH

This chapter explains our experiment approach. We use the experiment process given in Section 5.1 for defining, planning, and performing the experiment. The experiment definition is presented in Section 5.2 and the more detailed plan is presented in Section 5.3. In this section we also define the hypotheses and discuss the possible validity threats.

Refer to Appendix A for our definitions of concepts related to oil production.

5.1 Experiment process

Experiments are used when we want control over the situation and manipulate behaviour directly, precisely and systematically [24]. Proving that our agent system actually works would have to be done by deploying it in a real oil production system. This is of course not an option, and we will therefore put our agents to the test in a simulated oil production field. We are going to investigate a situation in which random selection and assignment are not possible, our experiment is therefore a *quasi experiment*.

An experiment process provides support in setting up and conducting an experiment. We will use the process suggested by [24], which consists of the following steps;

- **Experiment definition**
  Define the experiment in terms of problem, objective and goal.
- **Experiment planning**
  Determine context, state hypothesis, design experiment, and evaluate possible threats.
- **Experiment operation**
  Prepare the subjects and material needed, execute the experiment and collect the data.
- **Analysis and interpretation**
  Analyse and interpret the collected data.

The two first steps are documented in Section 5.2 and Section 5.3, respectively. The results of the experiment operation are presented in Chapter 9 and the analysis and interpretation of results can also be found there.

5.2 Experiment definition

Formulating the goal of an experiment is necessary in order to ensure that important aspects of the experiment are defined before the planning and execution take place. We use the Goal/Question/Metrics (GQM) template suggested by [24] to define our goal;

Analyse <Object(s) of study>
for the purpose of <Purpose>
with respect to their <Quality focus>
from the point of view of the <Perspective>
in the context of <Context>

The **objects of study** are the entities that are studied in the experiment. In our case, the agent system is the main object of study. It will be tested in a simulated environment and the results will be compared with the performance of a human operator in the same environment.

The **purpose** defines the intention of the experiment. We would like to evaluate the agent system's ability to control the production of an oil field.

The **quality focus** is the primary effect under study in the experiment. We will study the performance of the agent system with regards to how it deals with critical situations and the amount of oil produced under its control.

The **perspective** presents the viewpoint from which the experiment results are interpreted. We have chosen the perspective of a IT system researcher.

The **context** is the environment in which the experiment is run. It defines the personnel (subjects) and software artefacts (objects) involved in the experiment. In our system, a human operator (played by us) is the subject. The objects are our simulator and the JACK Intelligent Agents framework, both running on the Java Virtual Machine (JVM).

Our definition of the experiment goal can then be summarized as follows:

Analyse the agent system
for the purpose of evaluation
with respect to performance
from the point of view of IT system researchers
in the context of a simulated oil field

5.3 Experiment planning

The planning of an experiment prepares for how the experiment is conducted. We here present our context, the formal definition of our hypotheses, and our selection of variables. The experiment design is chosen based on the hypotheses and variables selected.

5.3.1 Context selection

The context of an experiment can be characterised according to four dimensions [24]. These dimensions are listed below together with an explanation of the characteristics of our experiment.

- **Offline vs. online**
  Our experiment is performed offline, i.e. the agent system is not deployed in a real oil field, but in a simulated oil production system.
- **Student vs. professional**
  The agent system's performance is compared with the performance of students acting as operators in a graphical user interface representing a very simplified control room.
- **Toy vs. real problems**
  The simulator does not aim on reproducing the complexity of a real system, but the main cause-effect relationships of the challenges we address are maintained.

- **Specific vs. general**
  The experiment deals specifically with control of oil production, but positive results may indicate software agents' applicability in similar domains.

5.3.2 Hypotheses formulation

A hypothesis is stated formally, and the data collected during the course of the experiment is used to, if possible, reject the hypothesis. Two hypotheses have to be formulated. The **null hypothesis** states that there are no real underlying trends or patterns in the experiment setting; the only reasons for differences in our observations are coincidental. The **alternative hypothesis** is the hypothesis in favor of which the null hypothesis is rejected [24]. We have formulated hypotheses for three aspects of the agent system's performance compared to the performance of a human operator.

The number of critical situations

H1.0 The introduction of the agent system will not lead to any changes in the number of critical situations encountered.

H1.1 The agent system will lead to a reduction in the number of critical situations.

The duration of critical situations

H2.0 The introduction of the agent system will not lead to any changes in the duration of critical situations.

H2.1 The agent system will lead to a reduction in the duration of critical situations.

The quantity of oil produced

H3.0 The introduction of the agent system will not lead to any changes in the amount of oil produced in normal situation.

H3.1 The agent system will lead to an increase in the amount of oil produced.

5.3.3 Variable selection

All variables that are manipulated and controlled are called independent variables. The variables that we want to study are called dependent variables. In our experiment, we want to study the effect of changing the system which controls the production of the simulated oil field. More precisely, we want to compare the performance of the agent system with the performance of a human operator. The control system is therefore a *factor*, i.e. an independent variable which is changed in order to see its effect. The agent system and human operator are *treatments* to the factor, i.e. particular values of the factor. The dependent variable is the performance of each control system with regards to increasing the oil production and avoiding/escaping critical situations.

5.3.4 Validity evaluation

It is important to consider the question of validity already in the planning phase in order to plan for adequate validity of the experiment results. The results should be valid for the population from which the sample is drawn and it may also be of interest to generalise the results to a broader population. The results are said to have adequate validity if they are valid for the population to which we would like to generalise [24]. We have identified the following validity threats:

- We will perform the testruns with only one dataset. For higher statistical power we should probably have run the tests several times with different datasets. For our purposes we believe that this is not necessary. We will therefore accept this threat and take the lack of statistical power into account when evaluating our results.
- The comparison of agent control versus manual control requires that the environmental influence on the production is identical for the two testruns. We will address this threat by generating values for the environmental variables in advance, and use the same dataset for both runs.
- We know the agent system's characteristics, and this might influence the way we construct the environment. We will address this threat by using a script that generates random values for the environment variables, rather than setting the values manually.
- The simulated oil production system and the user interface of the simulator are both very simple compared to real oil fields and real control rooms. Our results must be evaluated with this in mind and the operator testrun should only be used as a standard of reference for the evaluation of the agent testrun.
- The reliability of our results depend on the correct functioning of the simulator and whether or not we use objective measures. We will address these threats by focusing on testing during the development of the simulator and limit the use of metrics that involve human judgment.

5.3.5 Experiment design

An experiment design consists of a series of tests of the treatments [24]. Our experiment design consists of one factor (the control system) and two treatments (agent system and human operator) and we want to compare the two treatments. The simulator will also be run with no control system in order show the "natural" variation of the production, as well as demonstrating that the introduction of a control system will in fact improve the situation.

5.4 Experiment construction

The main goal of the agent system is to reduce the information load on the human operator. The desired result from achieving this is improved decision-making, leading to more optimised production with regards to the field's production targets. We have decided to measure the reduction of situations which in reality would require human interference, since this can be viewed as an indication of reduced load. Avoiding critical situations is important, but this should not lead to reduced oil production. The agents' ability to maximise the oil content of the production is therefore also an aspect we would like to measure.

Our agents are given a high degree of autonomy and they can perform their work without human interference. They are required to notify the operator when critical situations occur, and the operator may oversteer the agents if desired. However, for the purpose of separating the agents' performance from the operator's performance, the operator will not interfere with the agents' work when we run the tests.

We perform the following testruns:

- **No control system**
  The field runs without external control.
- **Operator control**
  The operator controls the production of the field.
- **Agent control**
  The agents control the production of the field.

The metrics we will use are the following:

- **Critical sand situations**
  - number of free-standing critical situations
  - duration of the longest-lasting critical situation
  - average amount of time spent in critical situations for each well
- **Critical water situations**
  - number of free-standing critical situations
  - duration of the longest-lasting critical situation
  - total amount of time spent in critical situations for the plant
- **Oil production**
  - amount of oil produced by the field in normal situation

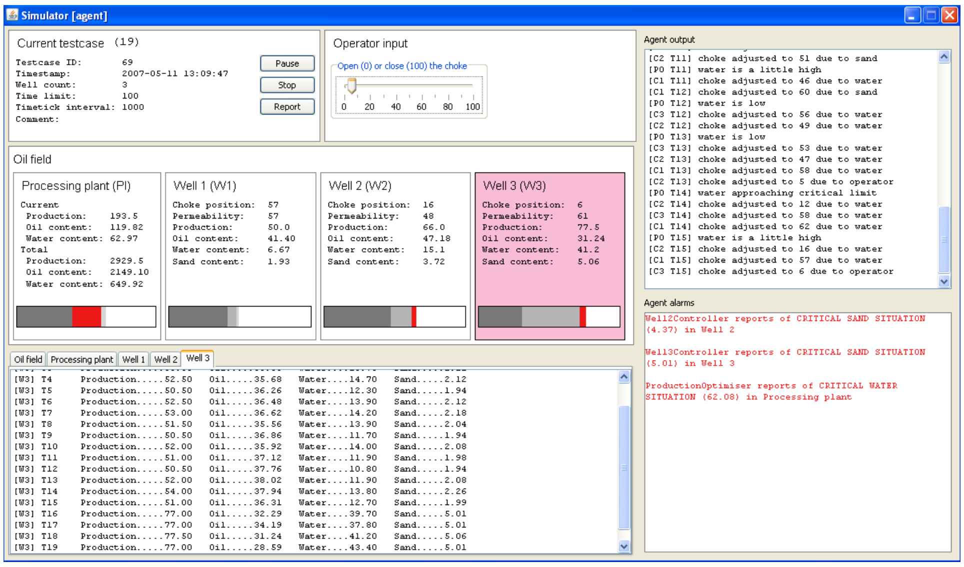
Chapter 6

Simulated environment

Our multiagent system has been designed for working in an oil field. In order to make sure that the agents work as intended, we develop a Java program which simulates the running of a simple field. This chapter presents the main features of the simulator, starting with a brief overview of its functionality in Section 6.1. Sections 6.2 and 6.3 describe how we define and calculate environmental variables and sensor values. A detailed description of how to run the simulator can be found in Appendix G.

6.1 Simulator overview

The simulator provides features which simplify the testing of the agent system, e.g. logging, agent output panels and report generation. The user interface of the simulator running in agent mode is shown in Figure G.3.



```
Testcase ID:        69
Timestamp:          2007-05-11 13:09:47
Well count:         3
Time limit:         100
Timetick interval:  1000
Comment:
```

```
Processing plant (PI)
Current
 Production:      193.5
 Oil content:     119.82
 Water content:   62.97
Total
 Production:      2929.5
 Oil content:     2149.10
 Water content:   649.92
```

```
Well 1 (W1)
Choke position:   57
Permeability:     57
Production:       50.0
Oil content:      41.40
Water content:    6.67
Sand content:     1.93
```

```
Well 2 (W2)
Choke position:   16
Permeability:     48
Production:       66.0
Oil content:      47.18
Water content:    15.1
Sand content:     3.72
```

```
Well 3 (W3)
Choke position:   6
Permeability:     61
Production:       77.5
Oil content:      31.24
Water content:    41.2
Sand content:     5.06
```

```
[W3] T4     Production.....52.50   Oil.....35.68   Water....14.70   Sand.....2.12
[W3] T5     Production.....50.50   Oil.....36.26   Water....12.30   Sand.....1.94
[W3] T6     Production.....52.50   Oil.....36.48   Water....13.90   Sand.....2.12
[W3] T7     Production.....53.00   Oil.....36.62   Water....14.20   Sand.....2.18
[W3] T8     Production.....51.50   Oil.....35.56   Water....13.90   Sand.....2.04
[W3] T9     Production.....50.50   Oil.....36.86   Water....11.70   Sand.....1.94
[W3] T10    Production.....52.00   Oil.....35.92   Water....14.00   Sand.....2.08
[W3] T11    Production.....51.00   Oil.....37.12   Water....11.90   Sand.....1.98
[W3] T12    Production.....50.50   Oil.....37.76   Water....10.80   Sand.....1.94
[W3] T13    Production.....52.00   Oil.....38.02   Water....11.90   Sand.....2.08
[W3] T14    Production.....54.00   Oil.....37.94   Water....13.80   Sand.....2.26
[W3] T15    Production.....51.00   Oil.....36.31   Water....12.70   Sand.....1.99
[W3] T16    Production.....77.00   Oil.....32.29   Water....39.70   Sand.....5.01
[W3] T17    Production.....77.00   Oil.....34.19   Water....37.80   Sand.....5.01
[W3] T18    Production.....77.50   Oil.....31.24   Water....41.20   Sand.....5.06
[W3] T19    Production.....77.00   Oil.....28.59   Water....43.40   Sand.....5.01
```

```
[C2 T11] choke adjusted to 51 due to sand
[PO T11] water is a little high
[C1 T11] choke adjusted to 46 due to water
[C1 T12] choke adjusted to 60 due to sand
[PO T12] water is low
[C3 T12] choke adjusted to 56 due to water
[C2 T12] choke adjusted to 49 due to water
[PO T13] water is low
[C3 T13] choke adjusted to 53 due to water
[C2 T13] choke adjusted to 47 due to water
[C1 T13] choke adjusted to 58 due to water
[C2 T13] choke adjusted to 5 due to operator
[PO T14] water approaching critical limit
[C2 T14] choke adjusted to 12 due to water
[C3 T14] choke adjusted to 58 due to water
[C1 T14] choke adjusted to 62 due to water
[PO T15] water is a little high
[C2 T15] choke adjusted to 16 due to water
[C1 T15] choke adjusted to 57 due to water
[C3 T15] choke adjusted to 6 due to operator
```

```
Well2Controller reports of CRITICAL SAND SITUATION
(4.37) in Well 2

Well3Controller reports of CRITICAL SAND SITUATION
(5.01) in Well 3

ProductionOptimiser reports of CRITICAL WATER
SITUATION (62.08) in Processing plant
```

Figure 6.1: Simulator in agent mode

The simulated field consists of a processing plant and a limited number of wells. Each well produce a mixture of oil, sand and water, and the processing plant receives the production from all the wells. The production of a well can be controlled by adjusting its choke. The valid choke positions range from 0 (open choke) to 100 (closed choke).

The methods used for defining the environmental influence on the production and calculating new sensor values are quite simple. They do not take into account all the dependencies found in a real system. The main point to make is that agents are in fact able to react dynamically to changes in the environment. To prove this, we only need values that vary in an unpredictable way, they do not necessarily have to be realistic. Adjusting the models in order to make them as realistic as possible has not been a priority.

Note that the values and ranges of our variables do not necessarily reflect reality. What we are primarily interested in is the relationship between them and their development over time. This also goes for the calculated sensor values.

6.2 Environment variables

The simulator generates time series for the values of environmental variables for each well. This can be done for every testrun, but it is also possible to use previously generated datasets. This enables us to run the system several times with the same dataset, e.g. for comparing the performance of the agent system with the performance of the operator in manual mode. An example of a dataset for a well is shown below;

```
#maxProduction
100
#timeticks
100
#permeability
57;57;55;55;55;56;57;56;57;56;57;56;57;56;...
#water ratio
0.128;0.119;0.113;0.100;0.113;0.100;0.100;...
#sand ratio
0.03198;0.02906;0.03206;0.02717;0.02432;...
```

`maxProduction` is the maximum possible production of the well, i.e. the production which may be obtained if the choke is fully open and the permeability is optimal. `timeticks` defines the duration of the testcase, and for each timetick, a permeability, water ratio and sand ratio are given.

Permeability, sand and water are defined as environment variables that differ for each well and vary over time. Permeability affects how freely the oil flows into the well. Low permeability means that the choke must be opened/closed more for the same effect to be achieved as the case would be with a higher value. Water ratio and sand ratio are variables that define the amount of water and sand in the production. The variation of these variables are defined in the following manner:

Permeability Each well is given a default permeability between 1 and 100, usually around 60. For each timetick, the permeability varies around its default value, usually within a bound of +/- 10. There is also a small probability for a sudden drop or jump in permeability, which may lead to values +/- 20 from the default.

Water ratio Each well is given a default water ratio between 0 and 1, usually around 0.3. For each timetick, the water ratio of a well typically changes with +/- 0.05 from the previous value. It might also be subject to sudden jumps or drops which may lead to values +/- 0.25 from the previous. There will always be at least ten percent water in the production of a well, but it will never exceed ninety percent.

Sand ratio Each well is given a default sand ratio between 0 and 1, usually around 0.025. The sand ratio varies over time and may change with +/-0.007 compared to the previous ratio. It might also be subject to sudden jumps or drops which may lead to values +/- 0.027 from the previous. There will always be some sand in the well, i.e. the sand ratio will never be zero, but the amount will never exceed ten percent of the production.

6.3 Sensor values

The sensor values for the wells are based on the environmental variables defined in the previous section. They are updated every timetick and calculated as follows:

Production The production rate of a well is calculated based on the well's maximum production rate and current permeability, as well as the current choke position.

Water The water sensor value denotes the current amount of water in the production. The water content is determined from the current water ratio and production rate.

Sand The sand sensor value denotes the current amount of sand in the production. The sand content is determined from the current sand ratio and production rate.

The sensor values of the plant are simply calculated as the sum of the corresponding sensor values of each well.

Chapter 7

System specification

The system specification is the initial phase of the Prometheus methodology. We start in Section 7.1 with a brief description of what our proposed system should be able to do. Section 7.2 describes the assumptions we make with regards to the availability of data and infrastructure. Section 7.3 contains the system goals. Based on them, a set of scenarios is developed and presented in Section 7.4. In Section 7.5 we describe the interface of the system, and the necessary roles are given in Section 7.6.

Refer to Appendix A for our definitions of concepts related to oil production.

7.1 System description

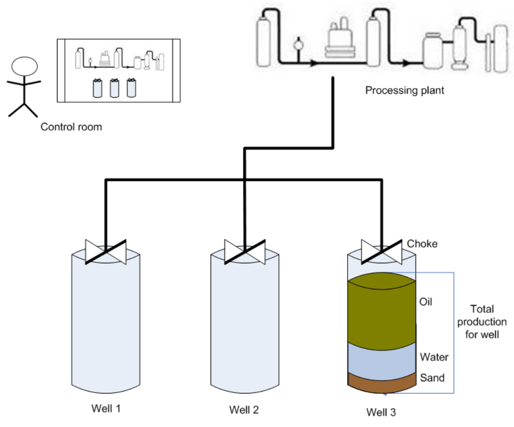


Figure 7.1: The agent system's environment

We would like to develop a distributed multiagent system which can perform analysis of sensor data and take actions based on its findings in order to optimise the production of a (simulated and simplified) oil field. The agents should be capable of adjusting the production of each individual well, taking into account local environment information. They should also be able to supervise a set of wells and take actions in order to maximise the production while at the same time ensuring that global restrictions are met. The agents control the production of an oil well by adjusting its choke (see Section 4.1.1). Optimisation is performed by maximising the total oil production while keeping its water content within a given limit. For each well it must also be assured that its production does not violate a given sand content limit. In addition to automatically regulating the oil production, the agent system should be able to communicate with the human operator of the control room. The operator should be notified when critical situations occur and the agent system should react and respond correctly to instructions from the operator.

The agent system's environment is illustrated by Figure 7.1.

7.2 Assumptions

Our proposed agent system is developed based on the simplified view of an oil production field which was described in Section 4.3. We make the following assumptions with regards to the agent system;

- The sensor data is cleansed and filtered by an external system before being made available for the software agents.
- The sand content limit for each well is known to the agent system.
- The water content limit for the processing plant is known to the agent system.
- The agent system has the necessary access for adjusting the chokes.
- The field is equipped with the necessary infrastructure for the software agents to be deployed distributedly.

7.3 System goals

The main goals of the agent system are briefly presented below, and their further decomposition is depicted in Figure 7.2.

[G1] Optimise oil production of field

The maximum allowed water content of the production is given by the amount of water that the processing plant is capable of handling. An evaluation of the current situation is done through analysis of sensor data. When optimisation is needed, the wells of the field are ranked according to how much water they produce, and optimal production rates for each well is calculated. From this, optimal choke positions for the individual wells are calculated and actuated.

[G2] Control sand content in well

In order to control the sand content of an individual well's oil production, the current sand content and the maximum limit must be known. Getting an overview of the current conditions in the oil well is done through analysis of sensor data. When the amount of sand indicates a need for production rate change, a new choke position is calculated and the choke is adjusted accordingly.

[G3] Perform operator's choke adjustment

The operator of the central control room should be able to oversteer the agent system if the need arises. The operator may request choke adjustments and the operator's request should always have priority.

[G4] Handle critical situation

The agent system should be able to detect and handle critical situations, i.e. situations which require human intervention. The agent system's main responsibility in such cases is to generate an appropriate alarm and notify the operator of the central control room.

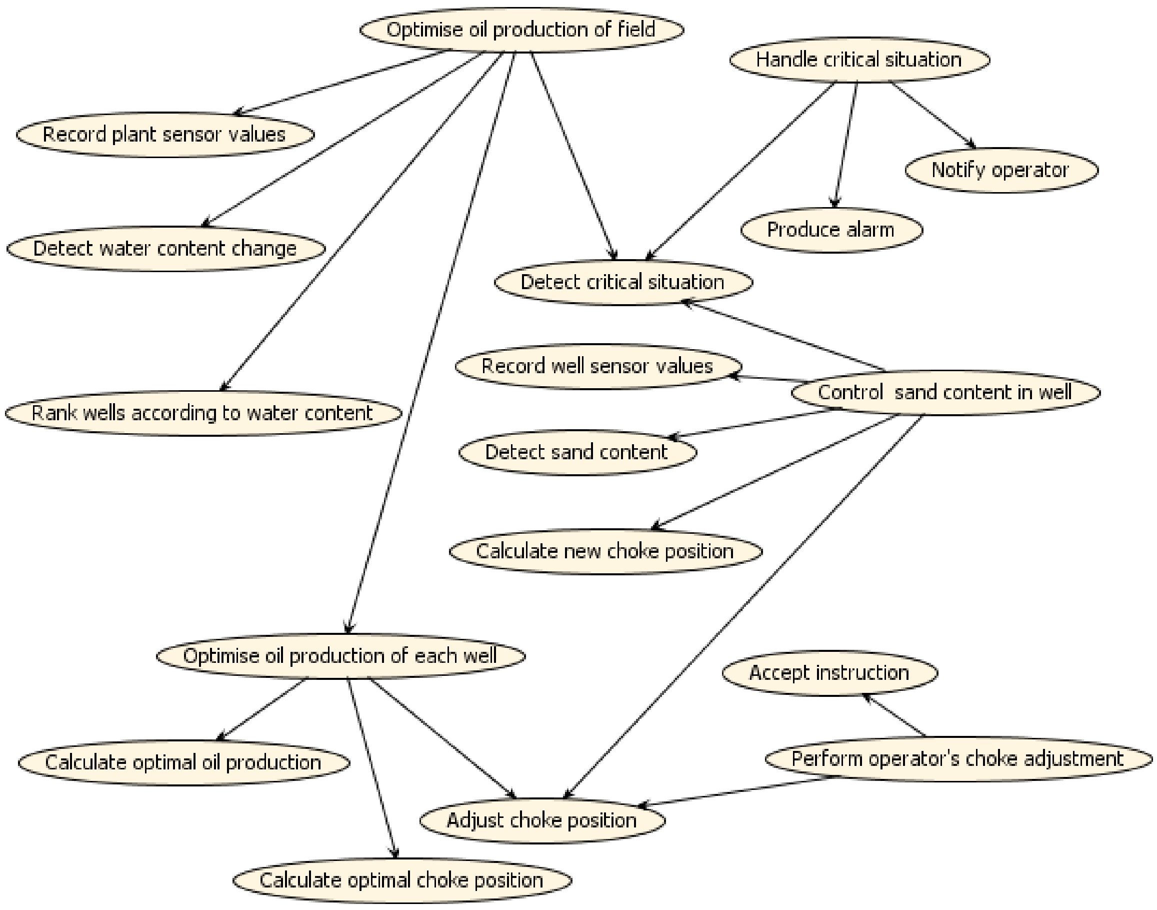

Optimise oil production of field
Handle critical situation
Record plant sensor values
Notify operator
Detect water content change
Produce alarm
Detect critical situation
Record well sensor values
Control sand content in well
Rank wells according to water content
Detect sand content
Calculate new choke position
Optimise oil production of each well
Accept instruction
Calculate optimal oil production
Perform operator's choke adjustment
Adjust choke position
Calculate optimal choke position

Figure 7.2: System goals

7.4 Scenarios

Here we describe the scenarios that take place in the system together with their appurtenant steps. The scenarios illustrate the normal running of the system. "OR" denotes situations where there are two alternative sequences. Figure 7.3 illustrate the scenarios and the connections between them.

[S1] Optimise oil production of field

Trigger: Change in water content of total production in the plant

The water content of the production is continuously monitored. When it changes, action is taken to optimise the production. In order to decide which chokes to adjust, the wells are ranked according to how much water they produce. During this process, critical situations should also be detected.

1. PERCEPT: Plant sensor values
2. GOAL: Record plant sensor values
3. GOAL: Detect water content change
4. GOAL: Detect critical situation
5. SCENARIO: S5

OR

5. GOAL: Rank wells according to water content
6. SCENARIO: S2

[S2] Optimise the oil production of each well

Trigger: Optimisation operation requested

When the optimisation process is initiated, the optimal production rates and choke positions for each of the required wells must be calculated.

1. GOAL: Calculate optimal oil production
2. GOAL: Calculate optimal choke position
3. SCENARIO: S4

[S3] Control sand content in well

Trigger: Change in sand content of production in the well

The sand content of the oil in each well is continuously monitored. When sand is detected, the choke position must be calculated and adjusted accordingly. Critical situations must also be detected.

1. PERCEPT: Well sensor values
2. GOAL: Record well sensor values
3. GOAL: Detect sand in oil production
4. GOAL: Detect critical situation
5. SCENARIO: S5

OR

5. GOAL: Calculate choke position
6. SCENARIO: S4

[S4] Adjust choke position

Trigger: New choke position requested or calculated.

When the system has recalculated the choke position or a request for a new choke position has been received, the change must be actuated.

1. ACTION: Change choke position

[S5] Handle critical situation

Trigger: Critical situation has occurred.

When a situation occurs that the system cannot handle without human intervention, an appropriate alarm must be created and sent to the central control room.

1. GOAL: Produce alarm
2. ACTION: Display alarm
3. GOAL: Notify operator

[S6] Respond to instruction from operator

Trigger: Choke adjustment received from the operator.

When the operator instructs the system to change a choke position, the system must perform the requested adjustments.

1. PERCEPT: Operator's choke adjustment
2. GOAL: Accept instruction
3. SCENARIO: S4

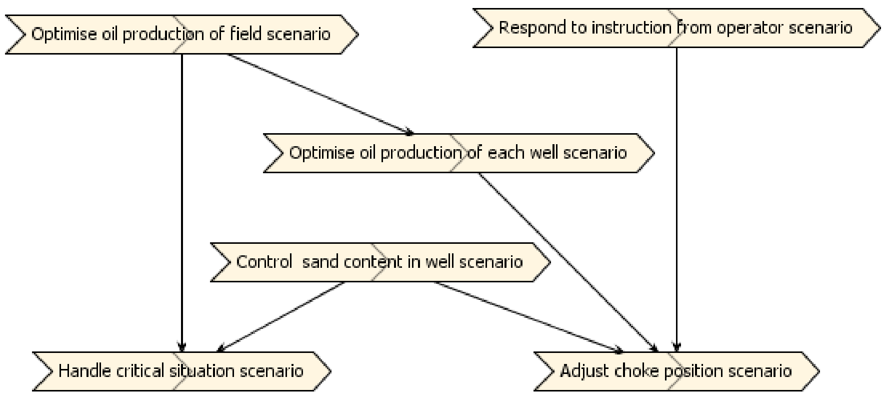

Optimise oil production of field scenario
Respond to instruction from operator scenario
Optimise oil production of each well scenario
Control sand content in well scenario
Handle critical situation scenario
Adjust choke position scenario

Figure 7.3: System scenarios

7.5 Interface

Through the definition of the scenarios we found the necessary percepts and actions.

The percepts are:

Well sensor values Sensor data is provided to the system at regular intervals. The data is saved as numerical values and analysed by the system. Analysis may involve comparison with earlier values in the database to determine how e.g. sand content is developing over time.

Plant sensor values This percept is analogous to the previous. Sensor data is stored as numerical values in a database and analysed by the system.

Operator's choke adjustment Choke adjustment instructions will be given by the operator to the system through a user interface. The instruction consists of a reference to a well and a numerical value denoting the new position of the choke.

The actions are:

Change choke position The system must be able to control the chokes of the wells. When a change in choke position is required, the new position must be given as input to the mechanics of the choke.

Display alarm An alarm containing relevant information must be displayed for the operator when a critical situation occurs. The information must include the affected component (one of the wells or the processing plant) and the sensor values that lead to the alarm being created.

7.6 Roles

The functionality of the system is described through roles. Each role can have goals, percepts and/or actions connected to them. The roles of the system are visualised in Figure 7.4.

[R1] Well monitoring

Trigger: New sensor values from well sensors

Description: Responsible for monitoring the production of a well in order to detect sand.

Goals: Record well sensor values, Detect sand content

Percepts: Well sensor values

[R2] Plant monitoring

Trigger: New sensor values from plant sensors

Description: Responsible for monitoring the production received in the production plant in order to detect changes in the water content.

Goals: Record plant sensor values, Detect water content change

Percepts: Plant sensor values

[R3] Choke position calculation

Trigger: Sand detected or automatic optimisation of well required

Description: Responsible for calculating new choke position when required. If new choke position is required due to sand in the well, a simple calculation following a few rules is performed. If new choke position is required in order to optimise the total production, a more sophisticated calculation is performed to find the optimal choke position.

Goals: Calculate new choke position, Calculate optimal choke position

[R4] Choke adjustment

Trigger: New choke position determined

Description: Responsible for adjusting the choke position when required.

Goals: Adjust choke position

Actions: Change choke position

[R5] Field optimisation

Trigger: Automatic optimisation required due to change in water content of the production in the processing plant

Description: Responsible for gathering water content information from all the wells and compare them to decide which wells should take measures to reduce or increase the production. Also responsible for calculating new production rate of the wells that should be adjusted.

Goals: Rank wells according to water content, Calculate optimal oil production

[R6] Critical situation detection

Trigger: Critical situation has occurred

Description: Responsible for detecting critical situations.

Goals: Detect critical situation.

[R7] Alarm creation

Trigger: Critical situation discovered

Description: Responsible for creating an alarm based on the available information about the critical situation.

Goals: Produce alarm

[R8] Alarm presentation

Trigger: New alarm created

Description: Responsible for notifying the central control room of critical situations that need attention.

Goals: Notify operator

Actions: Display alarm

[R9] Instruction acceptance

Trigger: Choke adjustment input from the operator
Description: Responsible for accepting instructions given by the operator.
Goals: Accept instructions
Percepts: Operator's choke adjustment

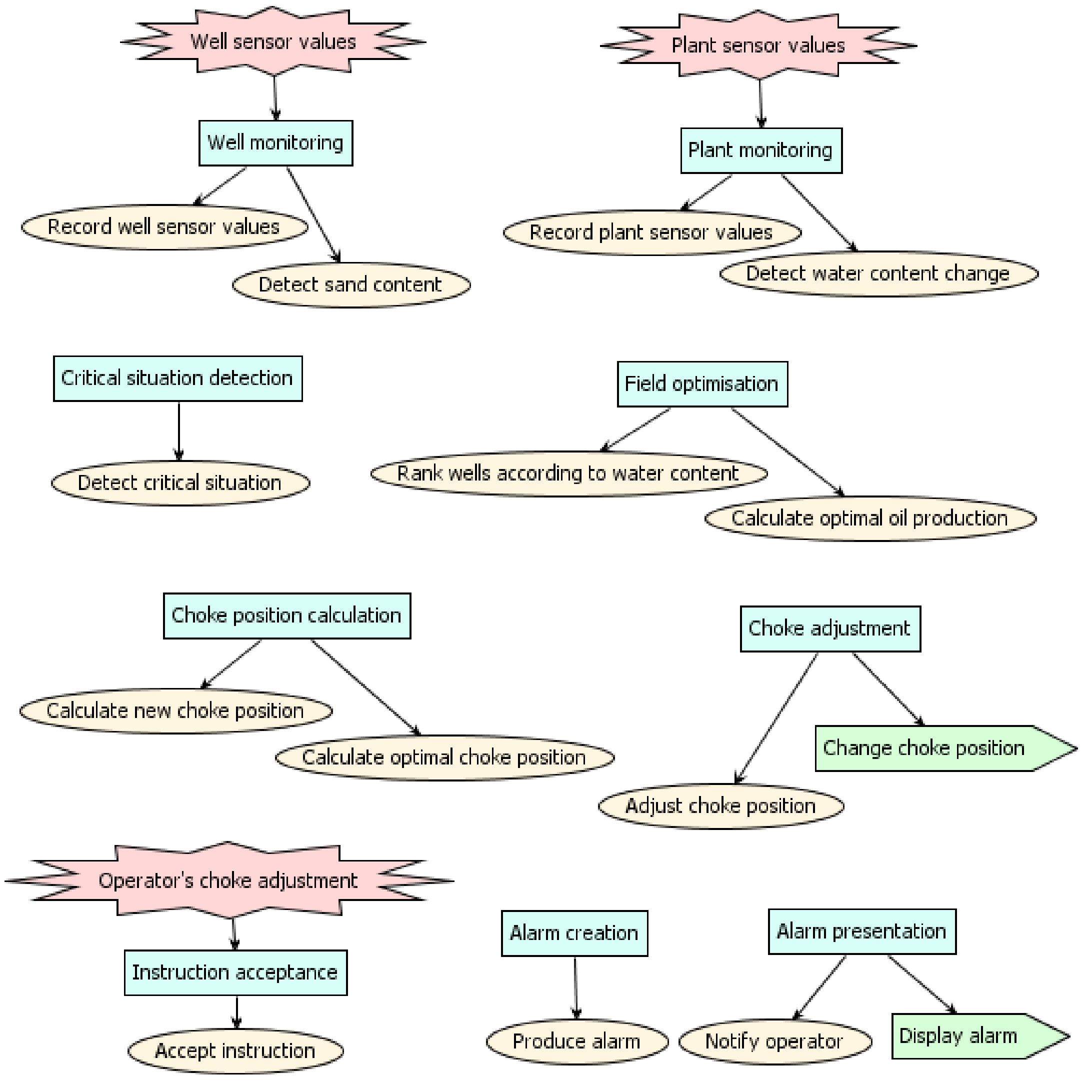

Well sensor values
Well monitoring
Record well sensor values
Detect sand content
Plant sensor values
Plant monitoring
Record plant sensor values
Detect water content change
Critical situation detection
Detect critical situation
Field optimisation
Rank wells according to water content
Calculate optimal oil production
Choke position calculation
Calculate new choke position
Calculate optimal choke position
Choke adjustment
Change choke position
Adjust choke position
Operator's choke adjustment
Instruction acceptance
Accept instruction
Alarm creation
Produce alarm
Alarm presentation
Notify operator
Display alarm

Figure 7.4: System roles

CHAPTER 8

SYSTEM DEVELOPMENT

The goal with this chapter is to define what agents are to be a part of the system, how they perform their tasks and how they interact with each other to meet the overall required functionality. The artefacts produced during the system specification are used as a basis for developing the high-level design of the agent system. Section 8.1 presents the architecture of our agent system. It starts with an illustration of how our agents are distributed in the simulated environment and continues with a detailed description of the agent types and the overall interaction between them. Section 8.2 takes a look at the details of the interaction and defines required messages and protocols. It also presents an example of the detailed design of an agent. Section 8.3 describes briefly how the design maps to JACK entities and Section 8.4 presents the agents' interaction with the simulated environment. The agent system's design is briefly evaluated in Section 8.5.

Refer to Appendix A for our definitions of concepts related to oil production.

8.1 System architecture

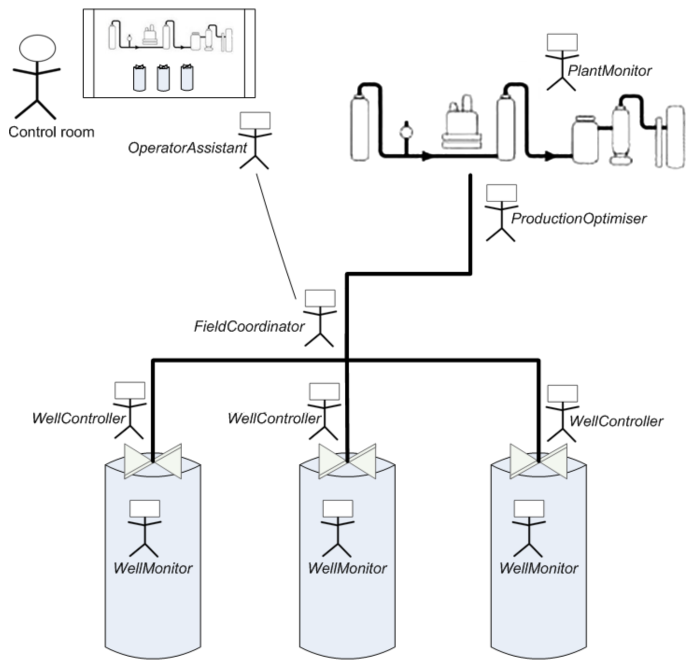


Figure 8.1: The agents in our simulated environment

In Section 4.3, we described our simulated oil production environment. Figure 8.1 illustrates how our agents fit into this environment.

Each well is equipped with an agent pair consisting of one `WellMonitor` and one `WellController`. Together, they monitor the well through analysing sensor values, and control the production of the well through adjusting the choke position. The plant is equipped with a `PlantMonitor` and a `ProductionOptimiser`. They monitor and analyse sensor values from the plant, and initiate an optimisation process when needed. The `FieldController` represents the whole field and its main responsibility is to work as a channel for the communication between the components of the field and the operator of the control room. The operator is represented by the `OperatorAssistant`, which is responsible for presenting output from the system to the operator and accepting input from the operator.

In the following sections, each agent type is described in more detail together with the overall interaction between them.

8.1.1 Agent types

The agent types of a system are defined by considering the roles and scenarios of the system specification. The agents should be evaluated against the criteria of coupling and cohesion, and determining the data needed by the different roles is of key importance here.

The nine roles which must be fulfilled by our system were given in Section 7.6 and are repeated for convenience here.

R1 *Well monitoring*

R2 *Plant monitoring*

R3 *Choke position calculation*

R4 *Choke adjustment*

R5 *Field optimisation*

R6 *Critical situation detection*

R7 *Alarm creation*

R8 *Alarm presentation*

R9 *Instruction acceptance*

The [R1] *Well monitoring* and [R2] *Plant monitoring* roles are responsible for updating the sensor value databases of the system. These databases contain data from all relevant sensors and the data is used by several roles; [R3] *Choke position calculation* is dependent on knowing the wells' production rates, [R6] *Critical situation detection* must have access to water and sand sensor values, and [R5] *Field optimisation* needs to know the water content of the individual wells and the processing plant. Role [R3] is also dependent on access to the Choke position database which is continuously updated by the [R4] *Choke adjustment* role. The roles [R7] *Alarm creation*, [R8] *Alarm presentation*, and [R9] *Instruction acceptance* are not directly dependent on any of the databases. Figure 8.2 illustrates the roles' use of the data stores just described.

We ended up with the following six agent types which fulfil the roles as illustrated by Figure 8.3.

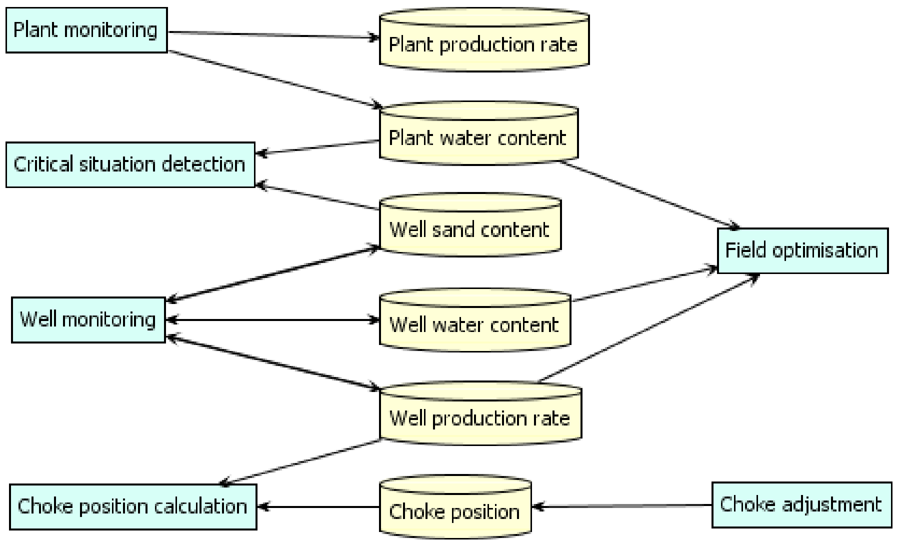


Figure 8.2: The data used by the roles

[T1] `WellMonitor`

Cardinality: One agent for each well

Lifetime: Lives as long as the well is in operation

Roles: [R1] Well monitoring

Description: The `WellMonitor` is responsible for continuously recording and analysing sensor data from the well. The goal of the analysis is to determine whether there is sand present in the production, and if so, notify the `WellController` of the situation.

Initialisation: The agent must be given information about the sensors it is supposed to monitor, as well as the name of the `WellController` with which it should communicate.

[T2] `PlantMonitor`

Cardinality: One agent for the processing plant

Lifetime: Lives as long as the processing plant is in operation

Roles: [R2] Plant monitoring

Description: The `PlantMonitor` is responsible for continuously recording and analysing sensor data from the processing plant. The goal of the analysis is to determine if the water content of the production is changing. Whenever the water content (change) passes a given limit, the agent notifies the `ProductionOptimiser`.

Initialisation: The agent must be given information about the sensors it is supposed to monitor, as well as the name of the `ProductionOptimiser` with which it should communicate.

[T3] `WellController`

Cardinality: One agent for each well

Lifetime: Lives as long as the well is in operation

Roles: [R3] Choke position calculation, [R4] Choke adjustment, [R6] Critical situation detection

Description: The `WellController`'s main responsibility is to control the choke of its well. In order to do this, it must first calculate the new choke position based on the desired change of the production. This calculation is performed when the `WellMonitor` reports of sand in the production, or when the `ProductionOptimiser` requests an adjustment of the well's production. The choke may also need to be adjusted if directly requested by the operator. The `WellController` is also responsible for detecting and reporting critical situations; here defined as dangerously high levels of sand in the production. In cases like these, the `WellController` must notify the `FieldCoordinator`.

Initialisation: The agent must be set in control of a well's choke. It also needs to know the name of the `WellMonitor` it should communicate with, the maximum production and the critical sand limit for the well it controls.

[T4] `ProductionOptimiser`

Cardinality: One agent for the processing plant.

Lifetime: Lives as long as the processing plant is in operation.

Roles: [R5] Field optimisation, [R6] Critical situation detection

Description: The `ProductionOptimiser`'s main responsibility is to initiate optimisation of the oil field when the `PlantMonitor` indicates that this is necessary. When this happens, the current water content values of all wells are collected and compared in order to determine which wells are in need of some readjustments. The agent is also responsible for detecting critical situations in the processing plant as well as reporting them to the `FieldCoordinator`. A critical situation is here defined as dangerously high levels of water in the production.

Initialisation: The agent needs the names of all the `WellControllers` it should communicate with during the optimisation process. It also needs to know the critical water limit for the processing plant.

[T5] `FieldCoordinator`

Cardinality: One agent that represents the oil field as a whole

Lifetime: Lives as long as the system is in operation

Roles: [R7] Alarm creation

Description: Responsible for creating alarms and passing them on to the `OperatorAssistant` whenever a `WellController` or `ProductionOptimiser` reports of a critical situation. On a lower level, this agent will also be responsible for forwarding messages between agents of the control room and the agents situated in the field.

Initialisation: The agent needs to know the names of the `OperatorAssistant` and all the field's `WellControllers`.

[T6] `OperatorAssistant`

Cardinality: One agent that represents and assists the operator

Lifetime: Lives as long as the system is in operation

Roles: [R8] Alarm presentation, [R9] Instruction acceptance

Description: The `OperatorAssistant` is responsible for accepting instructions from the operator and communicating them to the rest of the system. It is also responsible for presenting alarms generated because of a critical situation.

Initialisation: The agent needs to know the name of the `FieldCoordinator` and have access information to the operator's input device and the alarm display.

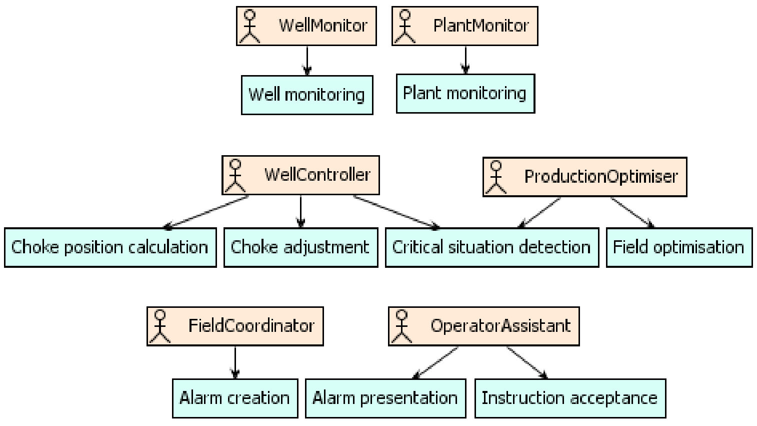


Figure 8.3: The agent types and the roles they fill

8.1.2 Agent interaction

The interaction between agents capture the dynamic aspects of the system. Figure 8.4 shows an acquaintance diagram that illustrates how the agents are connected. The `WellMonitor` sends messages to the `WellController` of the same well. The `PlantMonitor` communicates in the same way with the `ProductionOptimiser`. The `ProductionOptimiser` and the `WellControllers` communicate between themselves and the `FieldCoordinator`. The `FieldCoordinator` also communicates with the `OperatorAssistant`.

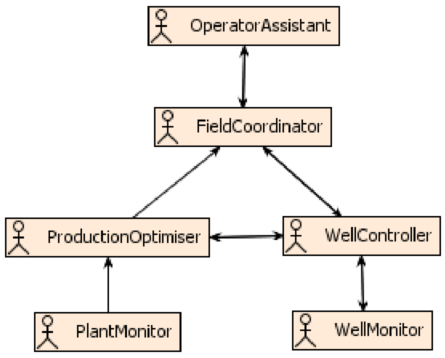


Figure 8.4: Agent acquaintance diagram

A more detailed description of the interaction is presented in Section 8.2.1.

8.2 Detailed design

In the beginning of this chapter we introduced Figure 8.1 which illustrated how the agents of our system fit into our environment. We have now described the roles and tasks of these agents, and the overall interaction between them. Figure 8.5 provides an overview of the architecture with the main entities involved. We will now take a closer look at the interaction, defining the necessary messages and protocols. We will also describe briefly how the internal design of an agent is defined.

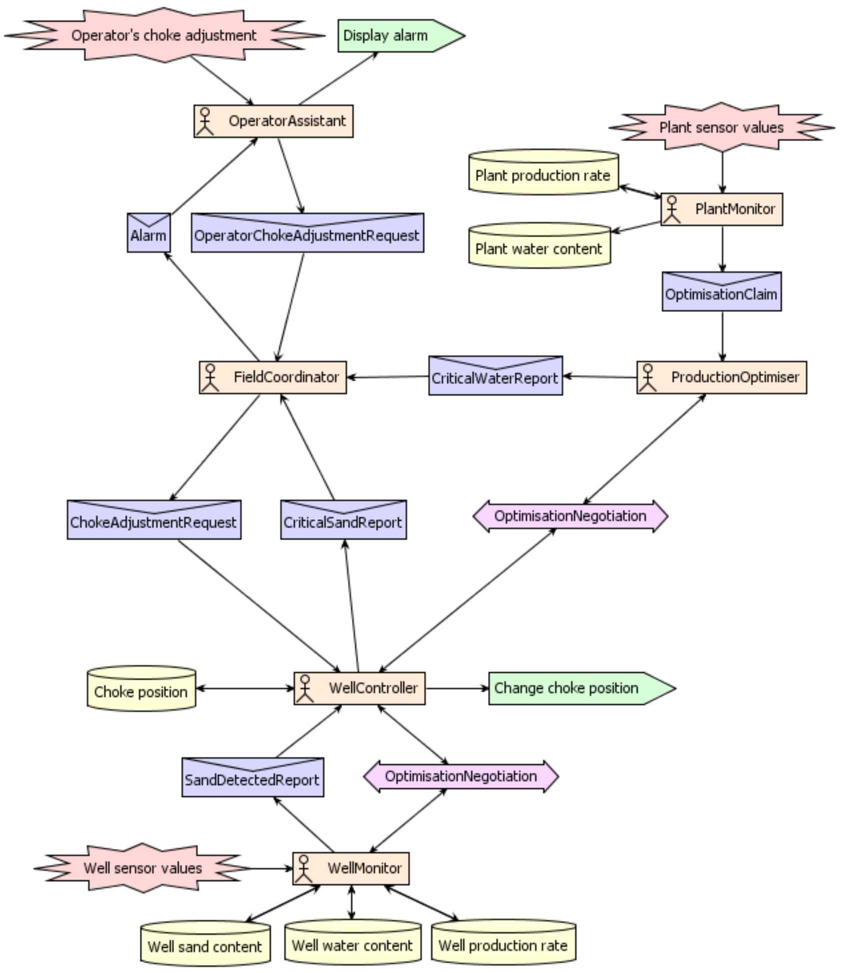


Figure 8.5: System overview

8.2.1 Interaction sequences

To illustrate the interaction taking place between the agents we set up interaction diagrams for the relevant scenarios from Section 7.4. Please note that arrows which starts or end in "nothing" indicates interaction with the agents' environment. The notation is borrowed from object-oriented design and is explained in Appendix B.3.

Scenario S1, S2 and S4

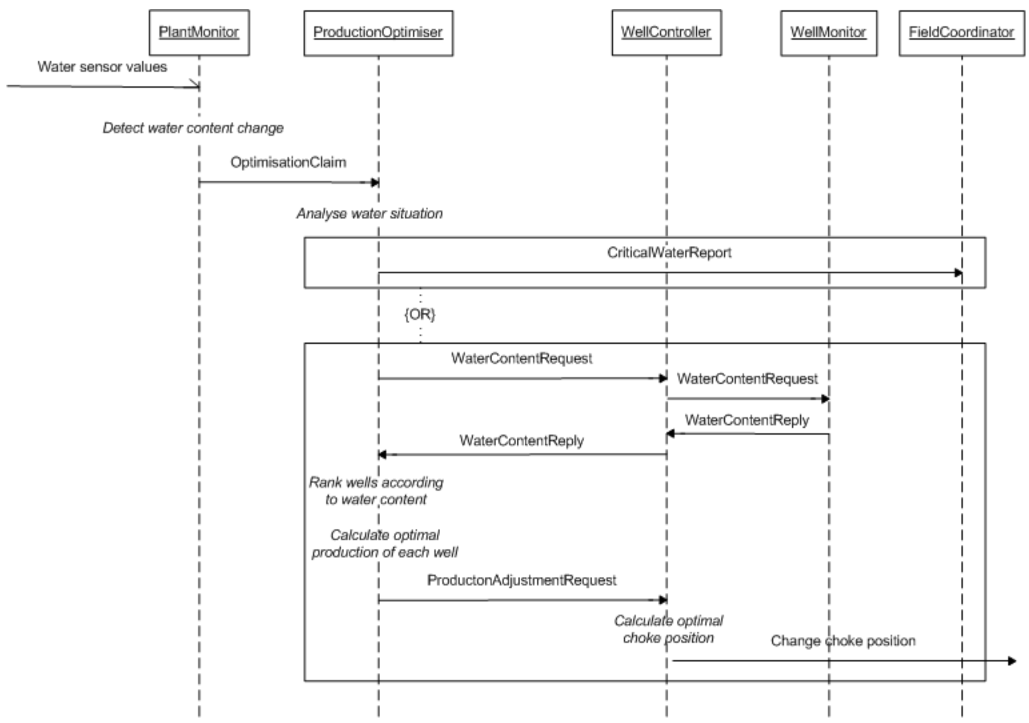


Figure 8.6: Interaction diagram for scenario S1, S2 and S4

Scenarios S1, S2 and S4 regard the steps taken by the system in order to optimise the oil production of the field. The `PlantMonitor` monitors the water content of the production in the processing plant continuously. When the water content (change) is above a given limit, an optimisation claim is sent to the `ProductionOptimiser`. The `ProductionOptimiser` decides whether the situation is critical, in which case it immediately sends a report to the `FieldCoordinator`, or requires production optimisation. The optimisation is performed by collecting the current water content values of each individual well, ranking the wells according to how much water they produce, and calculating the optimal production rate of each well. If this production differs from the current production, the `WellController` is asked to make the necessary adjustments of the choke. The `WellController` calculates the choke position required and changes the choke position accordingly. This interaction is depicted in Figure 8.6.

Scenario S3 and S4

Scenario S3 and S4 concern the way the system controls the sand content of each individual well, and the interaction is depicted in Figure 8.7. The `WellMonitor` continuously monitors the sand content of the production. When sand is detected, a report is sent to the `WellController` of the well. If the situation is critical, the `WellController` immediately sends a report to the

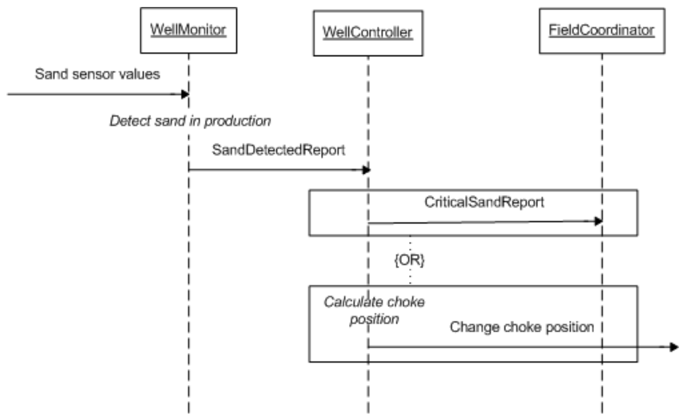


Figure 8.7: Interaction diagram for scenario S3 and S4

`FieldCoordinator`. If not, a new choke position is calculated and the choke is adjusted accordingly.

Scenario S5

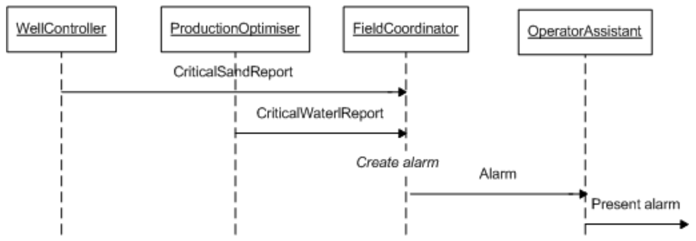


Figure 8.8: Interaction diagram for scenario S5

The interaction diagram for scenario S5 is shown in Figure 8.8. When a critical sand or water situation has been discovered elsewhere in the system, the `FieldCoordinator` is notified. Based on the available information, an alarm is created and passed on to the `OperatorAssistant` which is responsible for presenting it to the operator.

Scenario S4 and S6

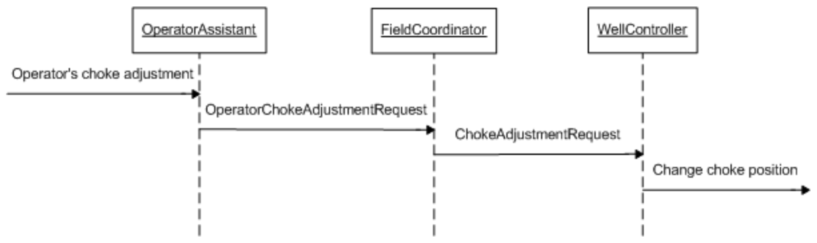


Figure 8.9: Interaction diagram for scenario S4 and S6

Scenario S4 and S6 concern the way in which the system deals with requests for choke adjustments from the operator of the control room. The `OperatorAssistant` is responsible for accepting the instruction and constructing an appropriate message for the `FieldCoordinator`. The request is then forwarded from the `FieldCoordinator` to the `WellController` which in turn makes sure the choke position is adjusted. This interaction is depicted in Figure 8.9.

8.2.2 Protocols

Interaction diagrams are meant to show a representative set of the valid sequences of interaction in the system. In order to have complete and precisely defined interactions, we have developed message descriptors and a protocol for the optimisation negotiation between the agents of the system. It is presented with descriptors and diagrams following the Agent UML protocol notation which is explained in Appendix B.2.

[P1] Optimisation negotiation

Description: The protocol for the negotiation between the `ProductionOptimiser`, `WellControllers` and `WellMonitors` during a production optimisation process. The `ProductionOptimiser` first collects the water content values from all `WellMonitors` (via the `WellControllers`) to rank the wells. From the ranking it decides which wells must adjust their production, calculates the optimal production rates, and instructs the corresponding `WellControllers` to perform the necessary adjustments. Figure 8.10 illustrates the protocol.

Agents: `ProductionOptimiser`, `WellController`, `WellMonitor`

Messages: WaterContentRequest (Table 8.1), WaterContentReply (Table 8.2) and ProductionAdjustmentRequest (Table 8.3)

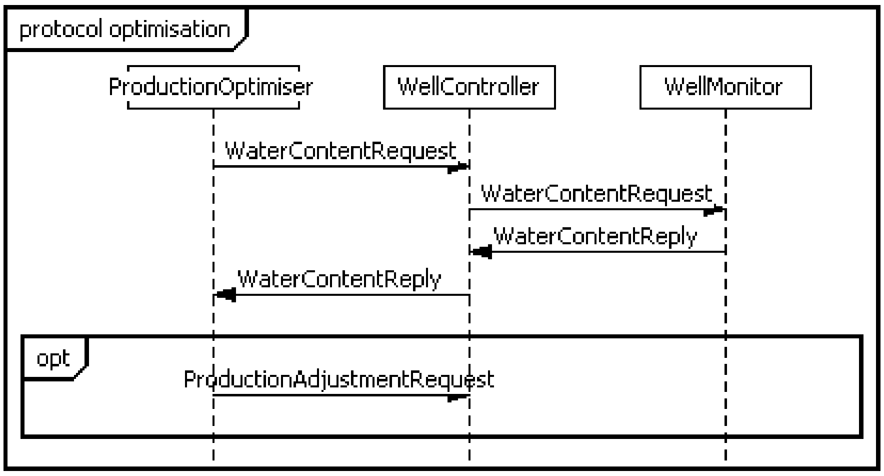


Figure 8.10: Negotiation protocol

[P1M1] `WaterContentRequest`	
Description:	A simple message which instructs the receiving agent to return the latest recorded water sensor value of the well. This is done whenever a need for optimisation of the field's oil production has arisen.
Sender:	`ProductionOptimiser`, `WellController`
Receiver:	`WellController`, `WellMonitor`
Information:	None

Table 8.1: [P1M1] `WaterContentRequest`

[P1M2] `WaterContentReply`	
Description:	The message sent as a reply to P1M1. The information contained in this message should enable the `ProductionOPtimiser` to calculate how the current production rate of each individual well should be regulated.
Sender:	`WellMonitor`, `WellController`
Receiver:	`WellController`, `ProductionOptimiser`
Information:	The current water content and production of the well.

Table 8.2: [P1M2] `WaterContentReply`

[P1M3] `ProductionAdjustmentRequest`	
Description:	This message is transmitted to the `WellControllers` of the wells which must regulate their production rate as part of the optimisation process. It contains the desired production rate for this specific well, as calculated based on the water content values of all the wells in the field.
Sender:	`ProductionOptimiser`
Receiver:	`WellController`
Information:	The new optimal production rate for this well together with the old (current) production and the reason for the request.

Table 8.3: [P1M3] `ProductionAdjustmentRequest`

8.2.3 Messages

Table 8.4 gives a list of the rest of the messages sent between the agents. The detailed descriptor tables referred to, can be found in Appendix E.

8.2.4 Internal agent design

The specification of functionality that we developed for the agents in Section 8.1.1 are combined to form capabilities and plans. Each agent is given a set of plans to achieve its required function, and some of the plans are grouped into capabilities. A plan is triggered either by an external message, a percept or an internal event. An internal event is usually a message sent by another plan within the same agent. An example of the internal design of an agent is given in Figure 8.11 which shows the internals of `WellController`.

`WellController` contains one capability and three plans, which together fulfils the agent's tasks; analysing the sand content situation, notifying the `FieldCoordinator` when the situation is critical, calculating and actuating new choke positions. The Figure illustrates how one plan can trigger another through the posting of an internal message. The blue-greyish messages are the external messages handled by the agent as defined in Sections 8.2.2 and 8.2.3.

Appendix C contains agent overview diagrams for the internals of all the agents in our system. It also contains a detailed description on how the agents perform their tasks through actuating their plans and sending and receiving the messages.

8.3 Implementation

Most of the design entities produced in the three main phases of the Prometheus methodology are carried through to the implementation of the system. JACK provides some entities which

Message	Agents involved	Descriptor
[M1] `OperatorChokeAdjustmentRequest`	From `OperatorAssistant` To `FieldCoordinator`	Table E.1
[M2] `ChokeAdjustmentRequest`	From `FieldCoordinator` To `WellController`	Table E.2
[M3] `OptimisationClaim`	From `PlantMonitor` To `ProductionOptimiser`	Table E.3
[M4] `SandDetectedReport`	From `WellMonitor` To `WellController`	Table E.4
[M5] `CriticalSandReport`	From `WellController` To `FieldCoordinator`	Table E.5
[M6] `CriticalWaterReport`	From `ProductionOptimiser` To `FieldCoordinator`	Table E.6
[M7] `Alarm`	From `FieldCoordinator` To `OperatorAssistant`	Table E.7

Table 8.4: External messages

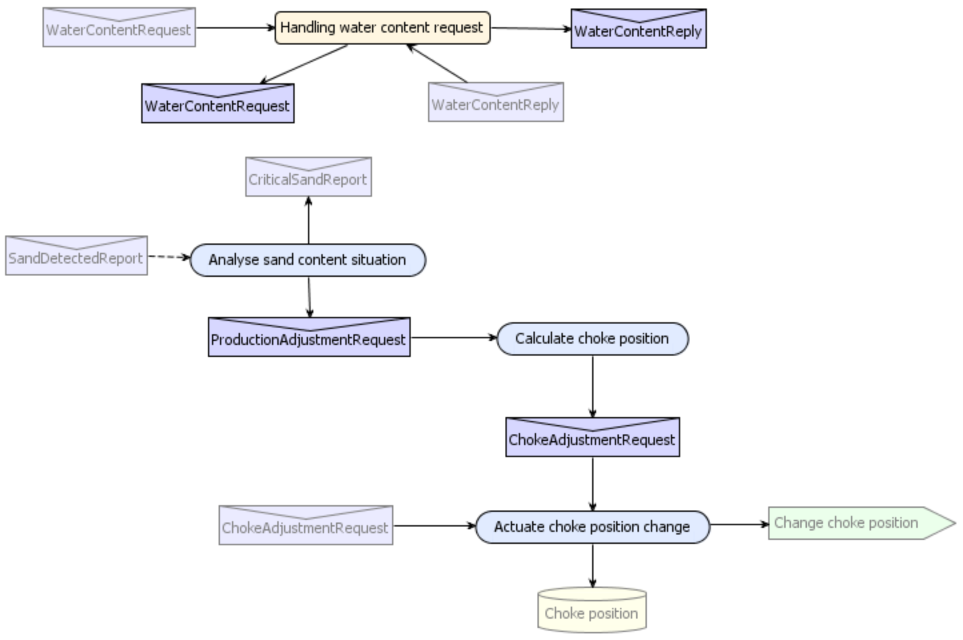


Figure 8.11: `WellController` overview

map directly to the corresponding Prometheus concepts (agents, capabilities, plans, beliefsets and events), and has mechanisms for representing concepts which are not directly implemented (percepts, actions and goals). Actions are, for instance, implemented as method calls or statements actuated inside the body of a plan.

Appendix D describes the implementation of our agent system. It includes an overview of the modifications made compared to the original design, and a more detailed explanation of our use of specific JACK entities in order to improve the design. Appendix F gives an overview of all JACK entities of our system. It also illustrates the changes made from design to implementation.

8.4 Agent/simulator interface

The interface between the agent system and the simulator is briefly explained below in terms of the agents' percepts and actions.

Agents' percepts The agents should be able to receive new sensor values from the wells and plant, and they must also be notified whenever the operator requests a choke adjustment. This is achieved through letting the `Sensors` class of the simulator extend the abstract Java class `Observable` and making the `WellMonitor` and `PlantMonitor` observers of sensor value changes. The same tactic is followed for the `OperatorAssistant` to be made aware of choke adjustment requests from the operator.

Agents' actions The agents need to control the choke of all wells and they must have the possibility of notifying the operator of critical situations in the field. Each `WellController` is therefore equipped with a reference to the `Choke` of their respective simulated well, and may adjust its position through calling the `changeChokePosition` method. The `OperatorAssistant` is responsible for displaying alarms from the agent system, and achieves this through a reference to the `AgentOutputPanel` of the simulator.

8.5 Evaluation

The benefits listed in Section 2.5 are exemplified in our agent system in the following manner. Agents are autonomous and can be trusted to pursue their goals. The WellController may therefore safely assume that all sensor values are recorded by the WellMonitor, and that it will be notified when the current sand content is too high. The ProductionOptimiser has a number of plans for dealing with high levels of water in the production. Its autonomy enables it to decide for itself which of the plans should be executed. This characteristic makes the agent system flexible, robust, and capable of adjusting to unpredictable conditions. The agents are situated in an environment which is affected by their choke adjustments, and which affects the agents through sensor values. The agents' social abilities enable them to cooperate in order to achieve a common goal. A good example of this is the optimisation process initiated when the water content suggests that this is needed.

Part III

Results and conclusions

CHAPTER 9

RESULTS

This chapter presents the results achieved through three testruns of the simulator. One in which there are no choke adjustments, another in which the operator controls the system, and a third where the agents are in charge. The first run is performed in order to provide basis of comparison for the other two testruns. The results are presented in Section 9.1 and Section 9.2 contains an analysis of the results with regards to the hypotheses defined in Chapter 5. The validity of the results is evaluated in Section 9.3.

9.1 Experiment operation

Experiment operation is the third step of the experiment process defined in Chapter 5. We now prepare and execute the experiment before we evaluate the results with regards to our hypotheses. We will focus on three main aspects, as defined by the hypotheses stated in Section 5.3.2:

The number of critical situations

The duration of critical situations

The quantity of oil produced

In Section 6.2, we explained how the dataset containing values for environment variables is generated and how sensor values are calculated based on these. We use identical datasets as input for the following three testruns;

Simulator run with no control system The agents are disabled and there is no operator input. The chokes are therefore never adjusted.

Operator testrun The agents are disabled and one of us acts as the operator. The operator adjusts the choke during the run with the goal of avoiding/escaping critical situations and maximising the amount of oil produced.

Agent testrun The agents control the chokes of the wells and adjust them to avoid/escape critical situations and maximise the oil production.

The duration of a testrun is 100 timeticks, each timetick lasts for one second. The field contains three oil wells and one processing plant. The default choke position is 50 and the maximum production of each well is 100. The critical water content limit of the plant is set to 70 and the critical sand content limit of each well is set to 3.

In the following section we present graphs which illustrate the sensor values' development over time for the plant and well 3, and tables which summarise the most important figures (rounded to one decimal place). The well graphs also include the choke position adjustments. The complete set of graphs for all three testruns can be found in Appendix H.

9.1.1 Simulator run with no control system

This run is without agents and without operator input. The graphs will therefore mainly illustrate how the sensor values vary "naturally" over time.

Figures 9.1 and 9.2 illustrate how the composition of the production changes over time for the processing plant and well 3. The graphs also show the critical limits for water in the processing plant and for sand in a well. Whenever the sand content of a well or the water content of the plant exceeds these limits, the situation is considered critical. We see that the natural variation in sand and water content leads to this several times. When critical situations occur, they remain critical until environmental changes normalise the situation.

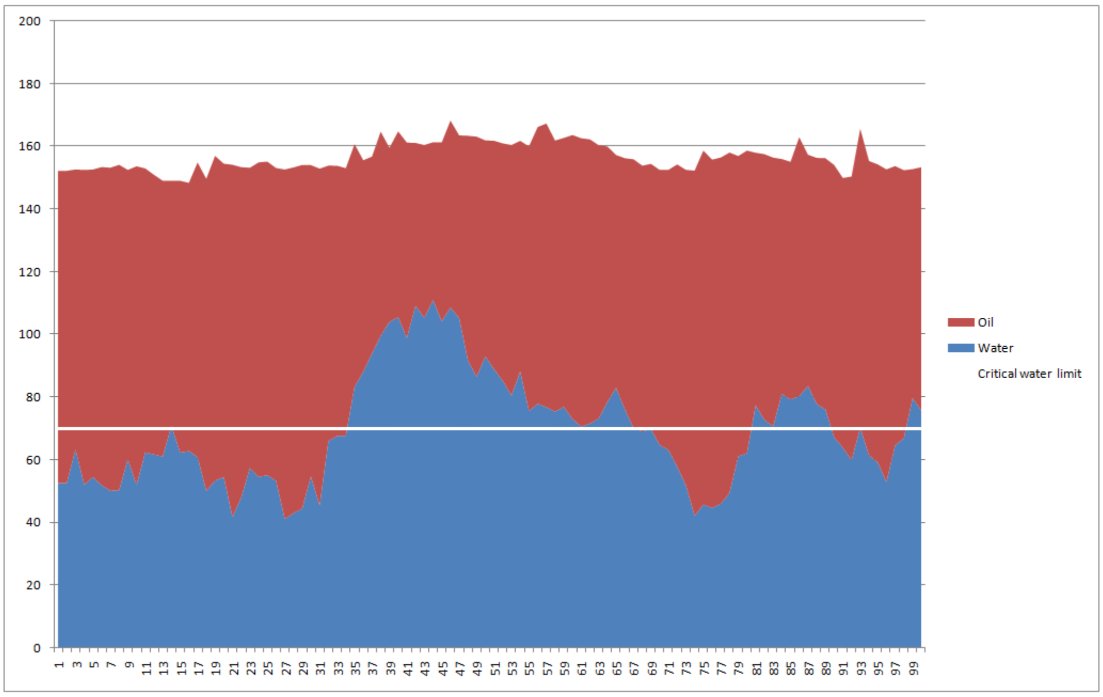


Figure 9.1: Simulator run with no control system - plant

9.1.1.1 Critical situations

Table 9.1 gives an overview of the critical situations that occur when the production is not controlled in any way. The number of free-standing critical situations refers to instants where the amount of water (for the plant) or sand (for the wells) exceeds the critical limit. There is only a limited number of free-standing critical situations in this run, but they last for quite some time. This results in the plant being in critical situation almost half of the running time.

	Well 1	Well 2	Well 3	Plant
Number of free-standing critical situations	1	2	5	4
Duration of longest-lasting critical situation	8	5	17	32
Total amount of time in critical situations	8	7	30	44
Largest amount of sand	3.6	3.5	4.2	-
Largest amount of water	-	-	-	110.8

Table 9.1: Simulator run with no control system - critical situations

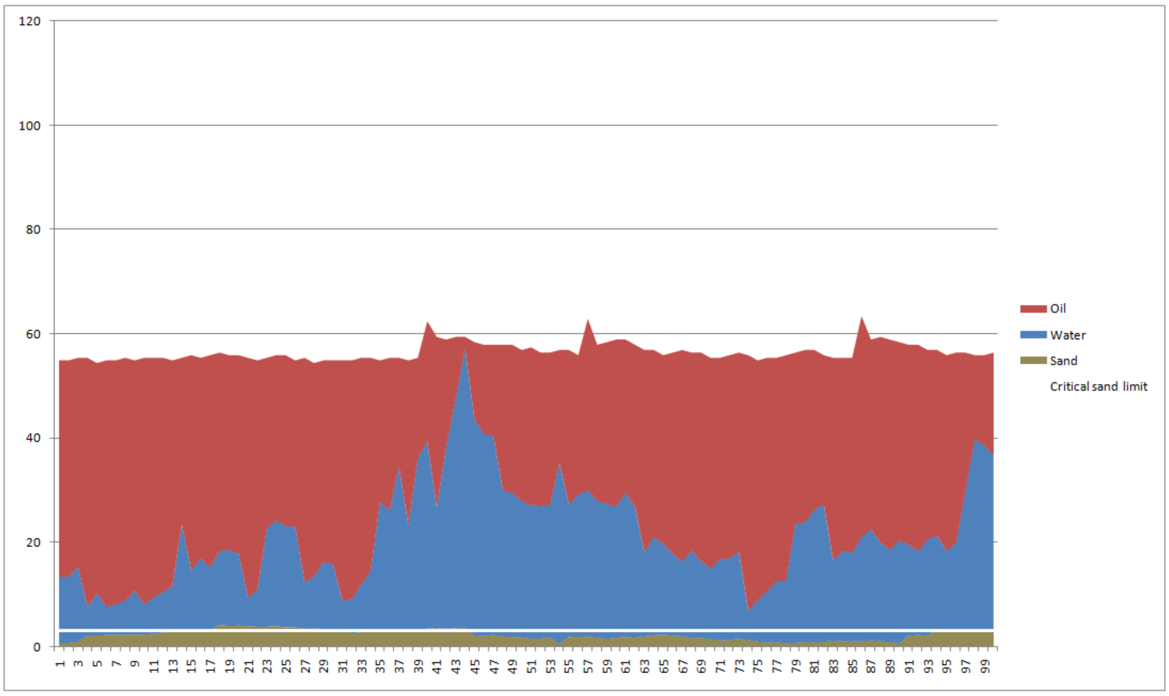


Figure 9.2: Simulator run with no control system - well 3

9.1.1.2 Production composition

Table 9.2 shows the amounts of oil, water and sand produced when the processing plant's water content is below the critical limit. It is worth noting that well 1 and well 3 contribute more than well 2 to the total production received in the plant. Well 2 appears to be producing a lot of water compared to the other two. The total quantity of oil produced is 5438.0.

	Well 1	Well 2	Well 3	Plant
Oil	2005.6	1185.3	2247.1	5438.0
Water	1011.2	1405.9	744.5	3161.6
Sand	107.7	94.8	134.4	336.9
Production	3124.5	2686.0	3126.0	8936.5

Table 9.2: Simulator run with no control system - production

9.1.2 Operator testrun

The simulator is run in manual mode with the agent system deactivated. The operator adjusts the chokes of the wells to avoid or escape critical situations and maximise the oil production.

Figures 9.3 and 9.4 illustrate how the composition of the production changes over time for the processing plant and well 3. Critical water situations are marked in the graph of the processing plant, and critical sand situations are marked in the graph of the well. The well's graph also includes the choke position as it is adjusted by the operator.

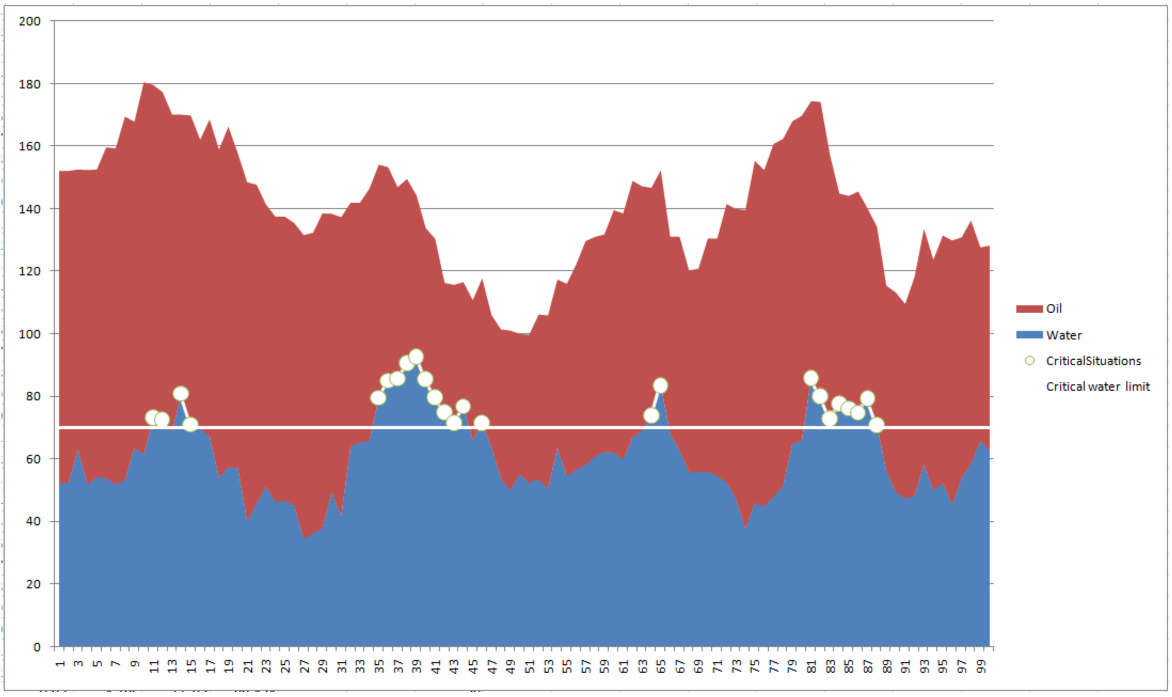


Figure 9.3: Operator testrun - plant

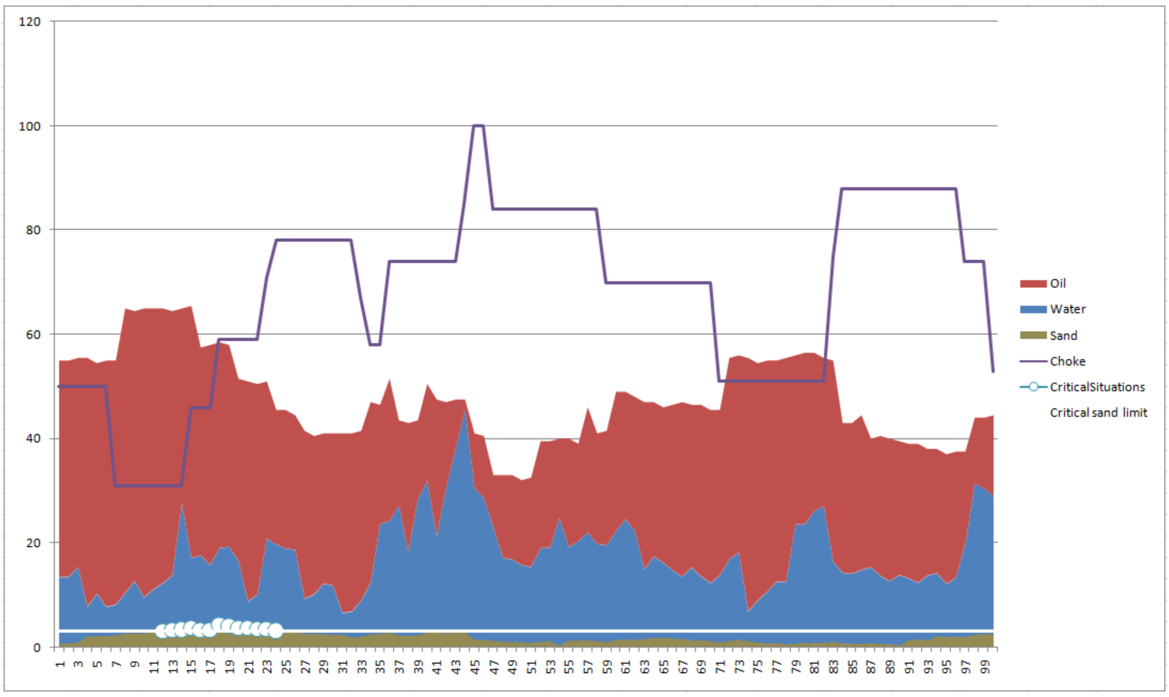


Figure 9.4: Operator testrun - well 3

9.1.2.1 Critical situations

As Table 9.3 shows, a total of ten critical situations occur in the wells and the processing plant during the testrun. The duration of the longest-lasting critical situation (13) indicates that it takes time for the operator to recognise critical situations and execute appropriate actions to stabilise the production. The operator's slow reaction is part of the reason why the largest observed amount of water is as high as 92.7 (22.7 above the critical limit).

	Well 1	Well 2	Well 3	Plant
Number of free-standing critical situations	1	2	1	6
Duration of longest-lasting critical situation	1	4	13	10
Total amount of time in critical situations	1	7	13	25
Largest amount of sand	3.1	4.0	4.3	-
Largest amount of water	-	-	-	92.7

Table 9.3: Operator testrun - critical situations

9.1.2.2 Production composition

Table 9.4 shows the amounts of oil, water and sand produced when the processing plant's water content is below the critical limit. The operator achieves an oil production of 6217.11.

	Well 1	Well 2	Well 3	Plant
Oil	2500.3	1370.5	2346.4	6217.1
Water	1230.4	1850.3	1043.8	4124.5
Sand	119.8	115.8	139.3	374.9
Production	3850.5	3336.6	3529.5	10716.5

Table 9.4: Operator testrun - production

9.1.3 Agent testrun

The simulator is run in agent mode. The agents continuously monitor the field's sensors, and adjust the chokes of the wells to avoid or escape critical situations while maximising the oil production.

Figures 9.5 and 9.6 illustrate how the composition of the production changes over time for the processing plant and well 3. Critical water situations and alarms are marked in the graph of the processing plant. No critical situations or alarms occur in well 3 during the agent testrun. The well's graph displays the choke position as it is adjusted by the agents, and we see that adjustments are made a lot more frequently than during the operator testrun.

9.1.3.1 Critical situations

As shown in Table 9.5, a total of eight critical situations occur in the wells and the processing plant during the testrun. The critical situations are all recognised immediately by the agents which notify the operator through alarms. The duration of the longest-lasting critical situation is two timeticks, indicating that the agents react quickly and are effective in stabilising the production.

9.1.3.2 Production composition

Table 9.6 shows the amounts of oil, water and sand produced when the processing plant's water content is below the critical limit. The agents achieves an oil production of 7334.98.

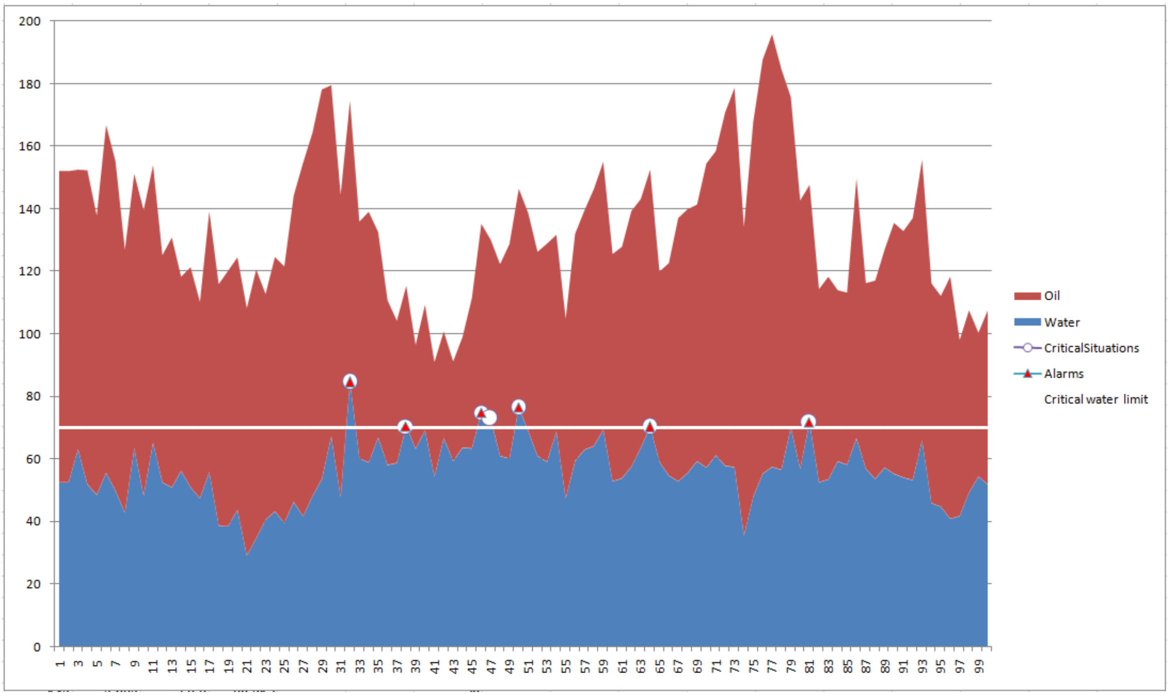


Figure 9.5: Agent testrun - plant

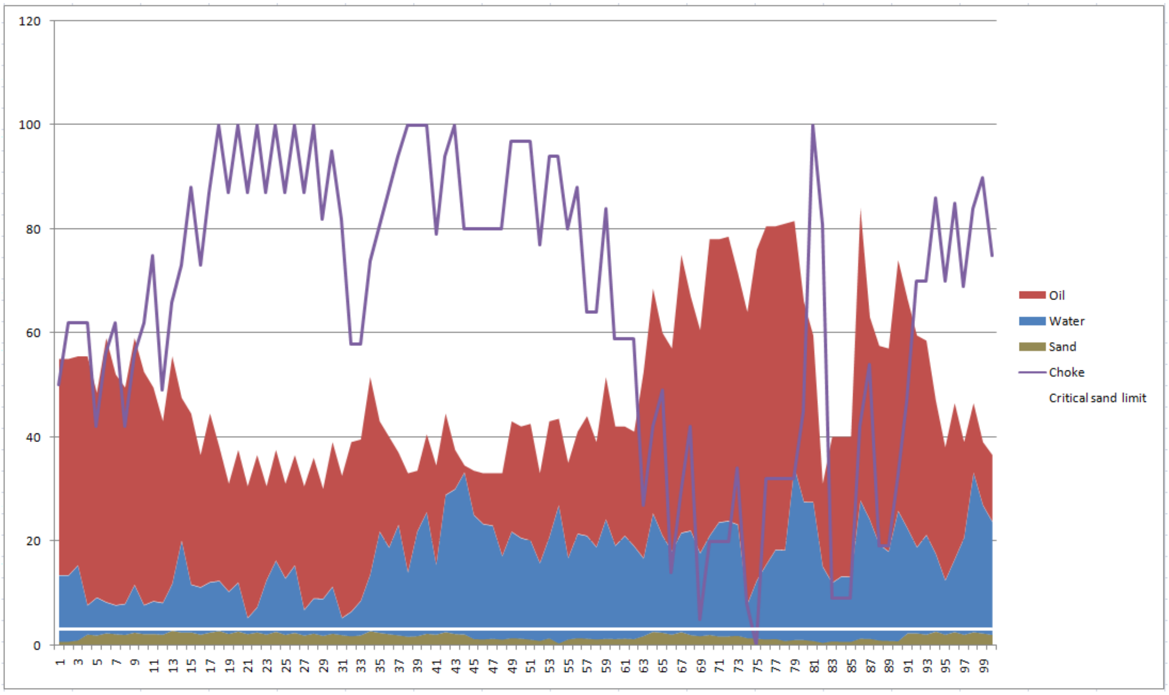


Figure 9.6: Agent testrun - well 3

	Well 1	Well 2	Well 3	Plant
Number of free-standing critical situations	2	0	0	6
Duration of longest-lasting critical situation	1	0	0	2
Total amount of time in critical situations	2	0	0	7
Number of alarms	2	0	0	6
Largest amount of sand	3.9	2.8	2.9	-
Largest amount of water	-	-	-	84.9

Table 9.5: Agent testrun - critical situations

	Well 1	Well 2	Well 3	Plant
Oil	2933.3	1452.7	2948.9	7335.0
Water	1646.1	1992.4	1437.9	5076.4
Sand	137.6	115.9	166.6	420.1
Production	4717.0	3561.0	4553.4	12831.5

Table 9.6: Agent testrun - production

9.2 Analysis and interpretation

This is the last step of the experiment process given in Chapter 5. Table 9.7 summarise the results of the different runs with regards to the metrics presented in Section 5.4.

	No input	**Operator testrun**	**Agent testrun**
Critical sand situations			
number of free-standing critical situations	8	4	2
duration of the longest-lasting critical situation	17	13	1
average amount of time spent in critical situations for each well	15	7	0,67
Critical water situations			
number of free-standing critical situations	4	6	6
duration of the longest-lasting critical situation	32	10	2
total amount of time spent in critical situations for the plant	44	25	7
Oil production			
amount of oil produced by the field in normal situation	5438.0	6217.1	7335.0

Table 9.7: Result summary

The results are now evaluated with regards to the hypotheses stated in Section 5.3.2.

9.2.1 The number of critical situations

The following hypotheses with regards to the performance of the agent system compared to the operator were formulated:

H1.0 The introduction of the agent system will not lead to any changes in the number of critical situations encountered.

H1.1 The agent system will lead to a reduction in the number of critical situations.

Table 9.7 shows that the total number of critical situations is reduced by both operator and agents. They both perform best with the critical sand situations, and do in fact increase the number of critical water situations. In total the operator reduces the number of critical situations with 16% and the agents reduce it with 33%. We therefore reject H1.0 and choose H1.1.

9.2.2 The duration of critical situations

The following hypotheses with regards to the performance of the agent system compared to the operator were formulated:

H2.0 The introduction of the agent system will not lead to any changes in the duration of critical situations.

H2.1 The agent system will lead to a reduction in the duration of critical situations.

The duration of the critical situations is reduced a lot. All metrics we have for this hypothesis show that the agents have a big effect on the amount of time the production is in critical state. The duration of the longest-lasting critical water situation is reduced with 69% by the operator and with as much as 94% by the agents. The duration of the longest-lasting critical sand situation is reduced with only 24% by the operator while the agents manage to reduce also this with 94%. The total amount of time in critical situation for the plant is reduced with 43% by the operator and with 84% by the agents. The average amount of time spent in critical situation for each well is reduced with 44% by the operator and is almost completely eliminated by the agents which reduce it with 96%. We therefore reject H2.0 and choose H2.1.

9.2.3 The quantity of oil produced

The following hypotheses with regards to the performance of the agent system compared to the operator were formulated:

H3.0 The introduction of the agent system will not lead to any changes in the amount of oil produced in normal situation.

H3.1 The agent system will lead to an increase in the amount of oil produced.

The quantity of oil produced when the operator is in charge is 14% higher than without a control system. The agents manage to increase the oil quantity with 35%. We reject H3.0 and choose H3.1.

9.2.4 Discussion

The graphical user interface of our simulated environment represents a very simplified version of a control room. In spite of its relative simplicity, it proved difficult to keep up with the production's development and handle critical situations during the operator testrun. Often, the wrong choke was adjusted or the adjustment was not adequate. The result was that several choke adjustments had to be made to stabilise the situation. The agents managed to keep the water content of the production more stable and closer to the critical limit, thereby achieving a higher oil production than the operator. They also made the right decisions on how to deal with critical situations and therefore stabilised the production immediately. Although the agent system did not achieve a big reduction in the number of critical situations, its reaction was correct and quick when they did occur. The amount of time spent in critical situations was therefore significantly reduced by the agents, also when compared to the operator's performance.

In a real production system, the number of wells that must be controlled is a lot higher than in our simulated environment. There are usually closer to a hundred wells, in some cases even thousands. Section 4.1.4 presented the Gullfaks field as an example. We also explained in Section 4.1.3 that optimisation today is often performed as rarely as on a monthly basis, weekly in the best cases. When we perform our testruns new sensor values are recorded every second. This does not happen in current systems, but real-time data is becoming more available and the desire for more frequent optimisation is there. The testruns show that an agent system is capable of working in a very unpredictable environment. Even though our simulator is far from a real production system, the results indicate that agents can perform better than humans when it comes to handling situations that demands quick reasoning and reaction. The agents perform their tasks distributedly and concurrently. Increasing the number of wells to control and restrictions to obey is therefore likely to affect the operator more than an agent system.

We believe our results suggest that an agent system could be introduced in a production system to deal with some aspects of the system control, thereby reducing the information load on the operator.

9.3 Evaluation

Validity is the strength of our conclusions, inferences and propositions. We here evaluate our results with regards to the validity threats identified in Section 5.3.4.

- We have accepted the threat of low statistical power. Our main goal has been to create a proof-of-concept and we believe that the testruns we have performed are adequate for demonstrating that agent systems may be suitable in this domain.
- A fair comparison of the testruns requires identical environmental influence on the production. The same dataset is used for all three testruns, thereby ensuring that this requirement is met.
- The environmental variables which influence the oil production are assigned random values generated by the simulator. This has been done in an attempt to avoid creating a dataset which is biased by our knowledge of how the agent system works.
- We have not drawn any conclusions with regards to a real operator's performance from the operator testrun. The operator's results are only used for the purpose of comparing a general manual solution with the performance of the agent system. Even though the simulator provides a restricted and simplified view of reality, we have tried to maintain the most important cause-effect relationships found in a real oil production system. We therefore believe that our results give indications of what can be achieved in a real-world scenario.
- The reliability of our results depends on the correct functioning of the simulator. We have addressed this threat by thoroughly testing that the values generated by the simulator agree with our manual calculations. The metrics we have used are objective and not dependent on human judgment.

CHAPTER 10

SUMMARY

Compared with other process industries, the oil and gas sector has historically been characterised by a fairly low level of automation. Recent years have seen an increase in the instrumentation of production facilities. The use of advanced sensor technology leads to a higher availability of real-time information about the state of the components and the processes taking place within them. The sensor data enables better control and optimisation of the oil production processes, but this requires that computer systems and control room operators are capable of quickly transforming raw data into knowledge and actions. Current technology is generally not able to perform the necessary intelligent reasoning without a great deal of human intervention.

Agent-oriented software engineering is a relatively new approach for designing, constructing and implementing computer systems. Adopting such an approach means decomposing the problem into multiple, autonomous components that can act and interact in flexible ways to achieve their objectives. Agents provide suitable abstractions for the creation of complex computer systems consisting of geographically dispersed components that exchange considerable amounts of data. These are the characteristics of a typical oil production scenario, where the wells are spread over a large area and the distances between the different installations

In a typical oil production scenario, the processes and equipment of the wells and processing plant are monitored, controlled and optimised from a centrally located control room. Optimisation of the oil production is usually performed manually and therefore not very often. A higher degree of automation is desired in the analysis of the production processes. In order to reduce the amount of data transmitted on the communication links, it is desirable to distribute this analysis so that reasoning is performed closer to the data sources. Rather than sending a continuous stream of sensor data to a central repository, entities situated near the sensors should analyse the data and only send reports which summarise their findings and actions. The human operator should only have to intervene when critical situations occur which cannot be dealt with automatically.

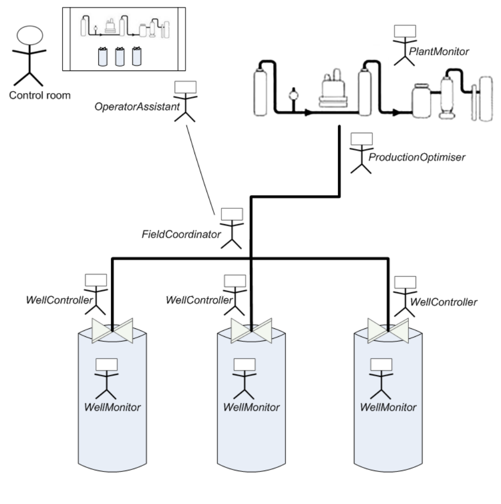


Figure 10.1: The agents in our simulated environment

We suggest a multiagent system for the control and optimisation of oil production as illustrated by Figure 10.1. In order to keep the complexity at a manageable level, a number of simplifying assumptions were made. We concentrated on the challenges related to limiting the content of water and sand in the production while maximising the amount of oil produced. We have considered sand a local restriction for the production of each well and water a global restriction for the production of the field as a whole. Our agents are distributed in the production system. Some of them are located at the wells, monitoring and analysing the wells' sensor values. Others are located at the processing plant, performing equivalent tasks there. These agents maintain an overview of the production content at all times. They know the critical limits of the wells and the plant and are therefore able to detect critical situations. Through adjusting the chokes of the wells, they try to maximise the oil production while keeping the system within its envelope. The system also includes two other agent types; one which represents the operator of the control room, another which represents all the agents of the field.

In order to test that the agent system works as intended, we developed an application simulating a simple oil field consisting of a limited number of wells and a processing plant. Controlling the production of a well can be done by adjusting its choke. The production is also affected by three relatively random variables; permeability, water ratio and sand ratio. The values of these variables were generated in advance, and this dataset was used for three testruns. Keeping the environmental influence on the production over time identical, our goal was to compare the effect of two different control systems: a human operator and our agent system. We also ran the simulator without a control system in order to obtain a standard of comparison.

Compared to the run without a control system, the operator and agent system both managed to reduce the amount of time spent in critical situations and increase the amount of oil produced. The achieved reduction in critical situations is higher for the agents than for the operator, and they also produce more oil. The results suggest that an agent system may be trusted to handle some of the tasks currently performed by human operators. The introduction of such a system may contribute to a reduction of the information load on the operator, presenting him/her with more time to concentrate on situations which the agents are not able (or not allowed) to handle on their own.

Chapter 11

Conclusion

The goal of this Master's thesis was to demonstrate the suitability of multiagent systems for control and optimisation of oil production. In order to achieve this, we implemented two systems. A multiagent system specifically designed for controlling the production of a set of oil wells, and a simulator providing an environment in which to test the agent system. Three hypotheses regarding the agent system's performance were formulated; we wanted to show that agents may be able to reduce the number of critical situations, reduce the duration of critical situations, and increase the amount of oil produced.

In spite of the comparative simplicity of the environment provided by the simulator, the main concepts of a real oil production system are in place. A set of wells produce a mixture of oil, water and sand which is delivered to a processing plant. The production of a well depends on varying environment variables and the current choke position. The water/oil and sand/oil ratios change constantly and in unpredictable ways. Controlling the production involves maximising the amount of oil, while at the same time making sure that a set of local and global restrictions are met. This might be performed manually, but quickly turns out complicated due to the amount of information available and the frequency with which it is updated.

The agent system is capable of continuously monitoring the wells and plant in our simulated environment. The agents base their actions on the analysis of real-time sensor data, thereby achieving an increase in the amount of oil produced while at the same time minimising the time spent in critical situations. The results of our testruns show that agents are able to work in an unpredictable environment, and perform similar tasks to those that are currently being done manually. The agents' performance matches or surpasses the performance of the operator in the simulated environment, indicating that an agent system could successfully be introduced in a control system to automate some of the operator tasks. This could result in a reduction of the information and work load on the operator and hopefully improve his/her decision-making.

Although our agent system performs very well in the given scenario, it is important to remember that agents are not the instant solution to all types of problems. Domains which are geographically distributed, complex, open and unpredictable are good candidates for the application of agent technology. Developing agent systems requires the same care which is exerted in traditional system development. An agent which is granted a high degree of autonomy must be thoroughly tested before deployment. Achieving the desired agent behaviour is not easy and may require a lot of tuning. In our opinion, the main advantage during system development is that the agent paradigm provides a very convenient level of abstraction. When deployed, a well-designed agent system provides very important features like flexibility, robustness, and maintainability - characteristics which fit perfectly with open and complex environments like oil production systems.

Our results led to the acceptance of all three hypotheses. We therefore believe that we have fulfilled our goal of demonstrating the suitability of agent technology in control and optimisation of oil production.

CHAPTER 12

FURTHER WORK

The problem area chosen as a basis for this thesis is interesting and very well suited for the application of agent technology. Our enthusiasm for the task at hand led to an initial system specification which was a lot more extensive than the design and implementation we ended up with. The main reasons for our simplifications were our virtually non-existent knowledge of the intricate details of the application domain, the time we had available and our limited experience with the design and implementation of agent systems. Although we were not able to follow all our ideas all the way through, we still believe that they are worth mentioning. This section contains some suggestions for the extension and improvement of the agent system we have designed.

12.1 Agent/operator cooperation

In our system, the software agents are mainly meant to alleviate the operator's work by assisting with the routine running of the system. Critical situations are dealt with by notifying the operator of the control room (and, for our testruns, choking the well which is believed to be causing the problem). One way of utilising the agents' capabilities even better would be to provide the operator with a suggested solution for the problem at hand. The alarm could consist of a sequence of steps meant for solving the critical situation. The operator could then choose to follow these steps or opt for an alternative solution to the problem. Whether or not the agents should in fact try to solve the problems on their own is a question of how much authority the operator is willing to surrender to an autonomous computer system. Achieving the right power balance is important for the successful adoption of such systems.

Our agent system is also capable of receiving input from the operator. There is currently only support for choke adjustment request, but it is easy to imagine other possibilities. The operator might want the agent system to provide an explanation of the reasoning leading to a certain decision. In critical situations there might be a need for an overview of the development of sensor values in a given well for a given time period. The agent system could quite easily be extended to support such features. The operator assistant agent is currently only responsible for accepting input and providing output. This agent could be given capabilities enabling it to handle other and more complex tasks, e.g. organising the user interface in order to emphasise the most important information at any given time.

12.2 Learning

Learning in agent systems is a very interesting topic which we would have loved to have the time to investigate further. This capability could be introduced at various places in our design and would, at least in theory, lead to a more dynamic agent system with the ability to adjust automatically to a changing environment.

We have identified the following areas in which learning could be successfully applied;

Environment monitoring The agents that are monitoring the development of environmental variables could be set to automatically learn what can be considered normal in their environment and what signifies a situation in need of attention.

Operator guidance In a system which provides the human operator with suggested solutions to problems that arises, learning would be of great use. Through analysis of the operator's response to the suggested solution, the system could learn which solutions are considered good and which are considered bad. This knowledge could then be applied when suggesting solutions for similar problems in the future.

Critical situation prediction If the agents could remember the sequence of events that happened previous to a critical situation, they could learn to recognise risky combinations of actions and environmental changes. This might give them the capability of predicting new critical situations, thereby being able to warn the operator in advance. This would enable the operator to take proactive measures to avoid such incidents, rather than reacting when they actually occur. For this to work as intended, it is important that the agents (to the extent possible) avoid giving such warnings for situations which do not turn out critical.

Choke adjustments Whenever the production of a well is adjusted through regulating its choke position, the actual effect on the environment should be observed and analysed. The findings from this analysis should be used to improve the reasoning methods of the agents so that the calculation of the optimal choke position is made more accurate over time..

12.3 "Real-world" testing

In order to test our agents, we made an application which simulates the running of a very simple oil field. We concentrated on a small number of variables in order to keep the complexity at a manageable level. We also used very simple models for how these variables develops over time. An obvious improvement of our system would be to use real-world data and more realistic models for the simulation of the oil production. One could also make a different selection of environment variables and include more than we have done. It would be interesting to modify the simulator in order to enable testruns with a much larger number of wells than the current maximum of four. Investigating the scalability would be of great importance since real oil fields may consists of hundreds of wells.

Part IV

Appendices

Appendix A

Terminology

Control system The person or system that is controlling the chokes of the wells. This is either the operator or the agent system. The goal of the control system is to maximise the production of oil while avoiding critical water situations in the plant and critical sand situations in the wells.

Critical sand situation A critical situation in a well is defined as a situation where the production of the well contains too much sand. High sand content levels have long-term negative effects (erosion on production assets), but no immediate effect on the production.

Critical water situation A critical situation in the plant is defined as a situation where the total production received contains too much water. High water content levels affect the production immediately (shut-down and loss of production).

Normal situation As long as the production is not in a critical water situation, it is defined as being in normal situation.

Oil field We refer to the collection of oil production components as the *oil field*. This includes the processing plant and the set of wells.

Oil well An oil well is viewed as a component which is capable of producing a mixture of fluids collectively referred to as *production*. The well has a maximum limit for how much sand it can safely produce. It contains sensors which may be read by external entities at any time.

Processing plant A processing plant is viewed as a component which is capable of receiving production. In our simulator, the processing plant does not actually process (transform) what it receives since this aspect is outside the scope of our work. The plant has a maximum limit for how much water it can handle. It contains sensors which may be read by external entities at any time.

Production The mixture of oil, water and sand which is produced by the oil wells.

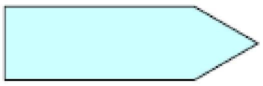
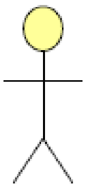
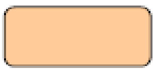
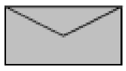
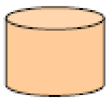
Appendix B

Notation

The notation used in the various design phases of the development of our multiagent system are explained in the following sections.

B.1 Prometheus Design Tool

An overview of the notation used for the diagrams in the Prometheus Design Tool is shown in Table B.1

	**Action** An action is what the agent does that effects the environment.
	**Agent** An agent receives percepts, sends and receives messages, reads and writes data and performs actions to obtain goals.
	**Capability** A capability is different roles of an agent grouped. It can be considered as a part of an agent.
	**Message** A message is data or request for data, and is sent by one agent to another.
	**Data** A data store is used to store beliefs of an agent.

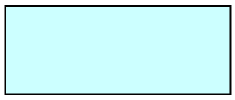
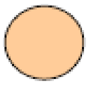
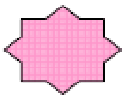
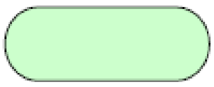
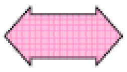
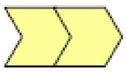
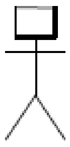
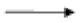
	**Role** The needed functionality of the system is grouped into roles that the agents inhabit.
	**Goal** A goal is something the agent should try to achieve.
	**Percept** A percept is the input coming from the environment to the agent.
	**Plan** A plan is a set of actions that an agent can perform to obtain a goal.
	**Protocol** A protocol is a specification of allowable agent interaction sequences, within a given conversation.
	**Scenario** A scenario is an abstract description of a particular sequence of steps within the system.
	**Actor** An actor is an entity (human or software/hardware) external to the system.
	**Edge** An edge is used to connect entities.

Table B.1: Prometheus diagram notation

B.2 Agent UML protocol

An overview of the notation used for the diagrams that describe the agent communication protocols can be found in Table B.2. The explanations are based on [23] and it is worth noting that the entities presented here is only a subset of the entities available for these types of diagrams.

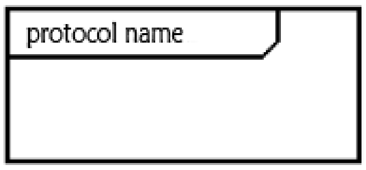
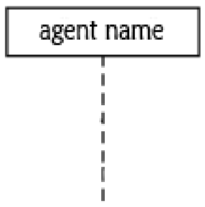
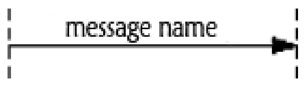
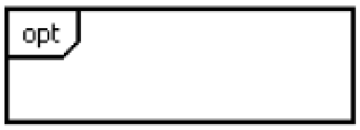
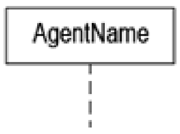
	**Protocol** The interaction protocol is enclosed in a box which is labelled with the protocol name.
	**Agent with lifeline** A protocol consists of a number of lifelines, each labelled with an agent name in a box at the top of the lifeline.
	**Message** Messages are depicted by labelled arrows between lifelines, and time increases down the page.
	**Option box** Option boxes may contain messages and other boxes, and denotes that the content may or may not occur.

Table B.2: Agent UML protocol notation

B.3 Interaction diagrams

Agent interaction diagrams are very similar to the sequence diagrams used in object-oriented design. An overview of the notation used for the interaction diagrams is shown in Table B.3

	**Agent with lifeline** An agent is shown as a box at the top of a dashed vertical line which represents the agent's life during the interaction.

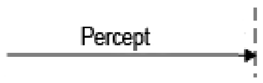
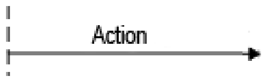
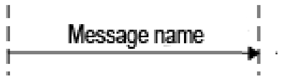
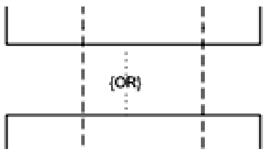
	**Percept** Input from the environment is shown as a horizontal arrow which starts in "nothing" and points towards the lifeline of an agent.
	**Action** Output to the environment is shown as a horizontal arrow which starts at the lifeline of an agent and ends in "nothing".
	**Message** Messages are shown as horizontal arrows which starts and ends in agents' lifelines.
	**OR** Alternative sequences are marked with *OR*

Table B.3: Interaction diagram notation

APPENDIX C

DETAILED DESIGN

In the following sections we present the internal design of our agents. The percepts and the internal messages are denoted Events and numbered *E<number>*. As the data used by the plans correspond to the agent's beliefs, they are from now on referred to as beliefsets.

C.1 PlantMonitor

The PlantMonitor's tasks is to record sensor values from the plant, analyse the water sensor values and notify the ProductionOptimiser when there is a change in water content. These tasks are fulfilled through the following plans:

P1 *Record sensor values from plant*

handles [E1] PlantSensorValues

sends [E2] NewWaterSensorValueNotification

P2 *Analyse water sensor values*

handles [E2] NewWaterSensorValueNotification

sends [M4] OptimisationClaim

P1 receives new sensor values through the event PlantSensorValues at regular time intervals, and adds them to the agent's set of beliefs. It then sends this in the internal message NewWaterSensorValueNotification to P2 for analysis. P2 compares the two water values to determine if there has been a change. If a change is discovered it creates an OptimisationClaim which is sent to the ProductionOptimiser.

Figure C.1 shows the internal design of the PlantMonitor.

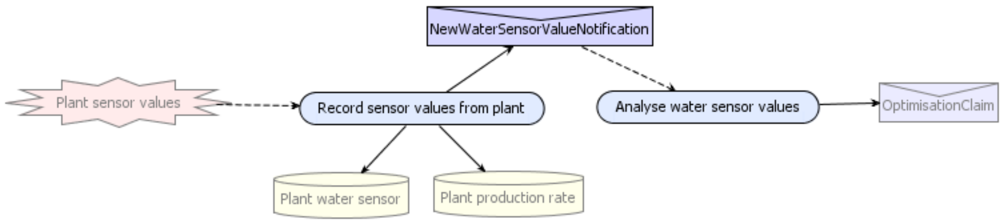

NewWaterSensorValueNotification
Plant sensor values
Record sensor values from plant
Analyse water sensor values
OptimisationClaim
Plant water sensor
Plant production rate

Figure C.1: PlantMonitor overview

C.2 `WellMonitor`

The `WellMonitor`'s tasks is to record sensor values from the well, analyse the sand sensor values and notify the `WellController` when sand is detected. This is fulfilled by the following plans:

P3 *Record sensor values from well*

handles [E1] `WellSensorValues`

sends [E2] `NewSandSensorValueNotification`

P4 *Analyse sand sensor values*

handles [E2] `NewSandSensorValueNotification`

sends [M4] `SandDetectedReport`

P5 *Look up water content*

handles [P1M1] `WaterContentRequest`

sends [P1M2] `WaterContentReply`

P3 receives new sensor values through the event `WellSensorValues` at regular time intervals, and adds them to the agent's beliefsets. It then sends a `NewSandSensorValueNotification`, containing the new sand sensor values, to P4 for analysis. P4 first checks if the new value is above a given limit, and if so, compares it with the previous in the database. If there has been a change in sand content a `SandDetectedReport` is sent to the `WellController`.

The `WellMonitor`'s plan P5 is initiated by a `WaterContentRequest` from the `WellController`. When such a request is received, P5 fetches the newest values from the well's water sensor and production rate beliefset. It wraps these values into a `WaterContentReply` which it sends back to the `WellController`.

Figure C.2 shows the internal design of the `WellMonitor`.

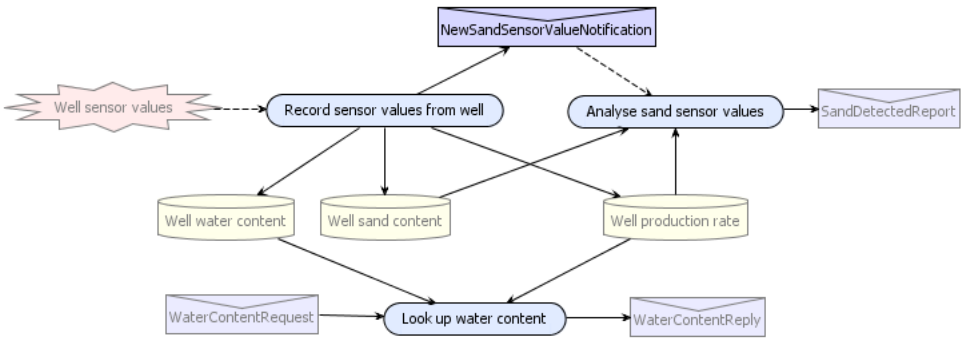


Figure C.2: `WellMonitor` overview

C.3 `WellController`

The `WellController` is responsible for analysing the situation when sand has been detected and notify the `FieldCoordinator` when the situation is critical. It is also responsible for calculating and actuating new choke positions whenever an adjustment is needed or requested. It fulfils its responsibilities through the following plans:

P6 *Analyse sand content situation*

handles [M4] `SandDetectedReport`
sends [M5] `CriticalSandReport`
sends [P1M3] `ProductionAdjustmentRequest`

P7 *Calculate choke position*

handles [P1M3] `ProductionAdjustmentRequest`
sends [M2] `ChokeAdjustmentRequest`

P8 *Actuate choke position change*

handles [M2] `ChokeAdjustmentRequest`

It also has the following capability:

C1 *Handling water content request* which contains the plans:

P9 *Forward water content request*
handles and sends [P1M1] `WaterContentRequest`

P10 *Forward water content reply*
handles and sends [P1M2] `WaterContentReply`

Plan P6 is initiated when a `SandDetectedReport` is received from the `WellMonitor`. The content is analysed by comparing the sand sensor value with various limits, taking into account if the well is currently in a critical state or not. If the value exceeds the critical limit, a `CriticalSandReport` is sent to the `FieldCoordinator`. If the value is below the critical limit, but above the limit for where action should be taken to avoid a critical situation, a `ProductionAdjustmentRequest` is sent to P7. The `ProductionAdjustmentRequest` contains a desired production rate which is calculated by P6 with consideration to whether the sand sensor value has increased or decreased.

Plan P7 can also be initiated by the same message sent by the `ProductionOptimiser`. P7 calculates the new choke position based on the production rate in the message. It sends the new position together with the reason for the adjustment to P8 in a `ChokeAdjustmentRequest`. P8 can also receive a `ChokeAdjustmentRequest` from the operator via the `FieldCoordinator`. If P8 receives requests from several agents at the same time, it knows which to prioritise. After determining if a requested change should be actuated, it stores the new choke position value in its choke position Beliefset and performs the change.

Figure C.3 shows the internal design of the `WellController`.

The C1 capability is a simple forwarding function. When the `ProductionOptimiser` needs to know the water content of the well, P9 receives the `WaterContentRequest` and forwards it to the `WellMonitor`. When the `WellMonitor` replies with a `WaterContentReply`, P10 forwards this to the `ProductionOptimiser`. Figure C.4 shows the internal design of *HandlingWaterContentRequest*.

C.4 `ProductionOptimiser`

The `ProductionOptimiser`'s tasks is to discover critical situations in the plant and to initiate optimisation when needed. It has the following plans:

P11 *Analyse water situation*

handles [M3] `OptimisationClaim`

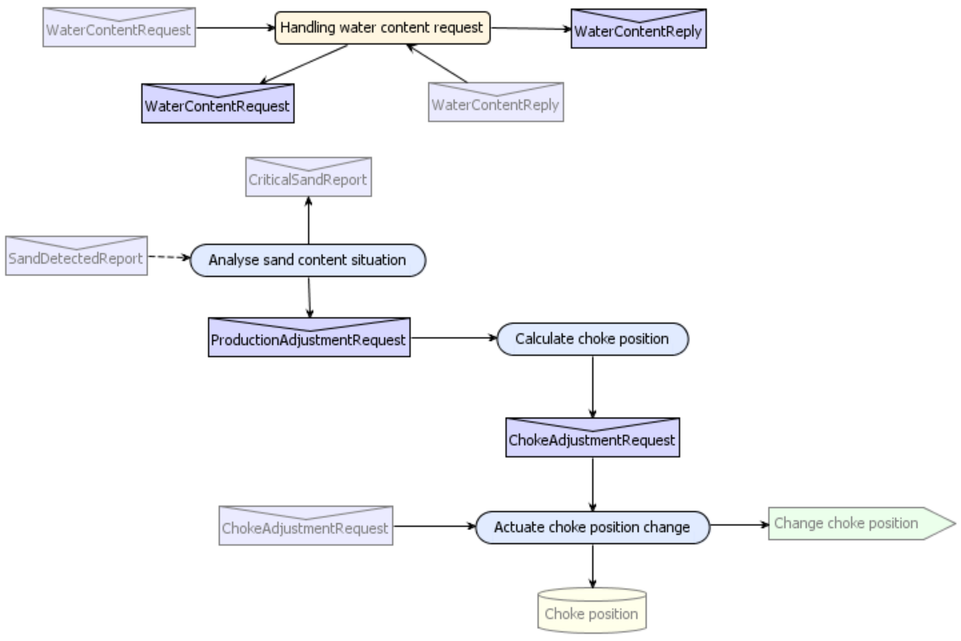


Figure C.3: `WellController` overview

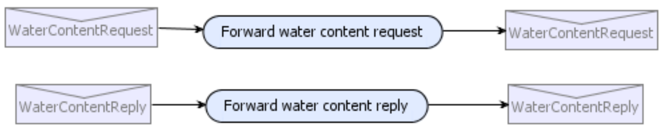


Figure C.4: `WellController` Capability [C1] *Handling water content request*

sends [M6] `CriticalWaterReport`
sends [E6] `FieldOptimisationRequest`

P12 *Calculate optimal production rates*

handles [E7] `RankedWellList`
sends [P1M3] `ProductionAdjustmentRequest`

It also has the following capability:

C2 *Ranking wells* which contains the plans:

P13 *Send water content request*
handles [E6] `FieldOptimisationRequest`
sends [P1M1] `WaterContentRequest`

P14 *Receive water content reply*
handles [P1M2] `WaterContentReply`
sends [E8] `UnrankedWellList`

P15 *Rank wells*
handles [E8] `UnrankedWellList`
sends [E7] `RankedWellList`

P11 is initiated by an `OptimisationClaim`. It analyses the water content by comparing the value with various limits and with the previous value. If the value is above the critical limit, a `CriticialWaterReport` is sent to the `FieldCoordinator`. If it is below the critical limit but there has been a change since the last value was recorded, a `FieldOptimisationRequest` containing the current water content value and the latest change is sent to C2. In C2, P13 handles the message. It initiates the task of collecting the current water content values of all wells by sending a `WaterContentRequest` to all `WellControllers`. P14 receives the replies and, when all replies have been gathered, initiates P15 by sending a `UnrankedWellList` containing the `WellControllers`' replies. P15 ranks the wells according to their water content and sends the rated list in a `RankedWellList` to P12. Depending on the situation, whether the water content is far from critical or approaching critical limit, P15 decides which of the wells to adjust and how much. It calculates the desired production rates and sends this to the relevant `WellControllers` in a `ProductionAdjustmentRequest`.

Figure C.5 show the internal design of the `ProductionOptimiser` agent, and Figure C.6 shows the internal design of the *Ranking wells* capability.

C.5 `FieldCoordinator`

The `FieldCoordinator` is the communication link between the field components and the control room. In this case it means that all messages between the `OperatorAssistant` and the other agents goes through the `FieldCoordinator`. It is also responsible for creating alarms when it is notified of critical situations. It has the following plans:

P16 *Forward choke adjustment request*

handles [M1] `OperatorChokeAdjustmentRequest`
sends [M2] `ChokeAdjustmentRequest`

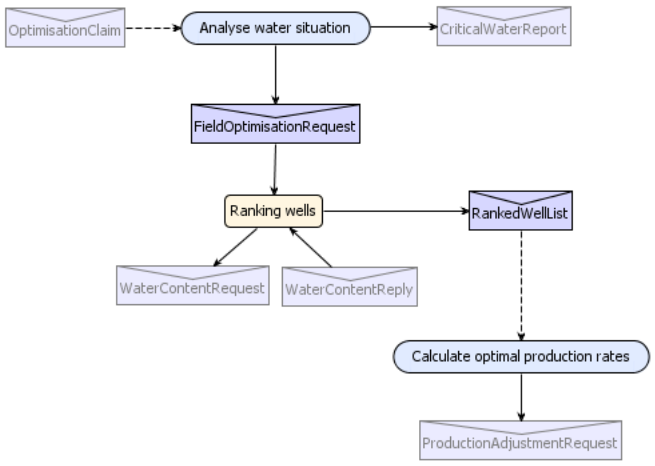


Figure C.5: `ProductionOptimiser` overview

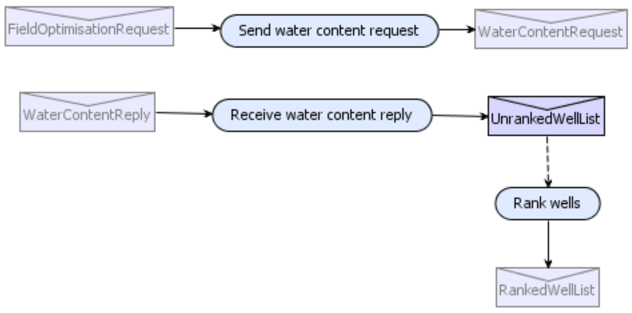


Figure C.6: `ProductionOptimiser` Capability [C2] *Ranking wells*

It also has the following capability:

C3 *Producing alarm* which contains the plans:

> **P17** *Create sand alarm*
>
> > handles [M5] `CriticalSandReport`
> > sends [M7] `Alarm`
>
> **P18** *Create water alarm*
>
> > handles [M6] `CriticalWaterReport`
> > sends [M7] `Alarm`

P16 receives `OperatorChokeAdjustmentRequest` from the `OperatorAssistant` and sends the relevant content in a `ChokeAdjustmentRequest` to the correct `WellController`.

C3 is responsible for producing alarms when a critical situation is discovered in the plant or in a well. The `WellControllers` send `CriticalSandReports` which P17 receives. P17 generates an `Alarm` from the information received, and sends it to the `OperatorAssistant`. Similarly, P18 receives `CriticalWaterReports` from the `ProductionOptimiser`, creates an alarm, and sends it to the `OperatorAssistant`.

Figure C.7 shows the internal design of the `FieldCoordinator` agent and Figure C.8 shows the internal design of the *Producing alarm* capability.

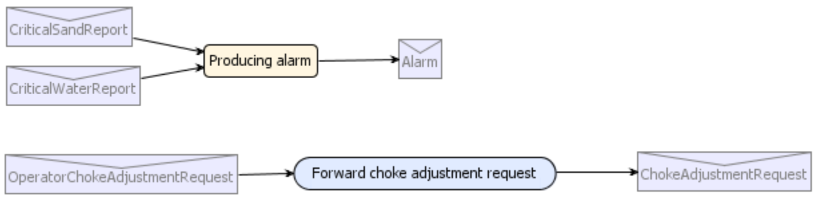


Figure C.7: `FieldCoordinator` overview

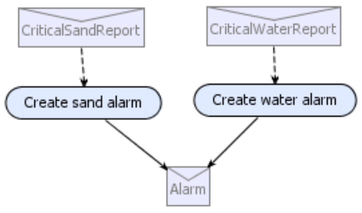


Figure C.8: `FieldCoordinator` Capability [C3] *Producing alarm*

C.6 `OperatorAssistant`

The `OperatorAssistant` is the link between the control room operators and the agents of the field. Its tasks includes accepting input from the operators and displaying system output (alarms). It has the following plans:

P19 *Accept operator's request*

> handles [E9] `OperatorsChokeAdjustment`

sends [M1] `OperatorChokeAdjustmentRequest`

P20 *Present alarm*

handles [M7] `Alarm`

P19 accepts requests for choke position changes from the operator (received through the event `OperatorsChokeAdjustment`) and sends the request in an `OperatorChokeAdjustmentRequest` to the `FieldCoordinator`. P20 is responsible for notifying the operator of critical situations by displaying the `Alarms` sent by the `FieldCoordinator`.

Figure C.9 show the internal design of the `OperatorAssistant`.

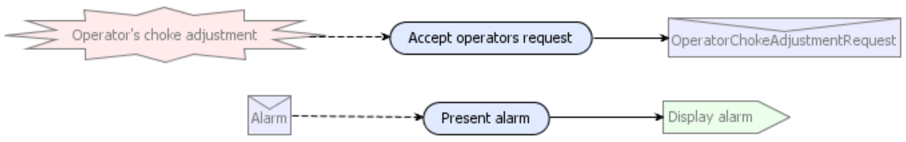


Figure C.9: `OperatorAssistant` overview

Appendix D

Implementation

In this appendix we explain the actual implementation of the agent system. To make the most out of the possibilities offered by JACK, some changes were made to the design during implementation. Most of these changes involve splitting up existing plans, and grouping these into capabilities. Section D.1 gives an overview of the modifications made. Our use of special JACK concepts like BDIGoalEvent and Semaphore is explained in Sections D.2 and D.3.

D.1 Design modifications

In the detailed design phase, one plan was made for each desired functionality of an agent, and capabilities were created only when it was very obvious that a functionality could have two distinct inputs (which called for two distinct plans, performing the same task but with different inputs) or when a collection of plans seemed to be especially connected. During implementation we found that some of our plans were unnecessarily complex, with several possible subtasks to perform. The selection of subtask was based on the outcome of a small analysis of the input values. We therefore decided to improve the design by dividing these plans into several plans, using the context method defined in Jack.

The context method *specifies a logical condition that must be satisfied if the plan is to be applicable for handling a given event in the current situation* [6]. This way, the analysis of input values is done in the context rather than in the body of the plan, thereby deciding which *plan* to perform instead of which subtask. Thus the plans become more consistent and clear.

Plans were modified in the `WellController`, `ProductionOptimiser` and `FieldCoordinator`. This section summarises the modifications made for each of these agents.

D.1.1 `WellController`

The plan `AnalyseSandContentSituation` was split into two new plans which were given a parenting capability *Analysing sand content situation*. The two new plans are `AnalyseSandContentCriticalSituation` and `AnalyseSandContentNormalSituation`. The original plan evaluated current situation and performed different subtasks depending on whether the situation was critical or not. The two new plans each handles one of those situations.

Listing D.1 shows how the context method of `AnalyseSandContentCriticalSituation` checks whether the current sand content is above the critical limit of the well (the limit is known by the agent).

```
public plan AnalyseSandContentCriticalSituation extends Plan {
   #uses interface WellController self;
   #handles event SandDetectedReport sanddetectedreport0;
   #sends event CriticalSandReport criticalsandreport1;
   static boolean relevant(agents.field.well.SandDetectedReport ev) {
      return ev.currentSand!=null;
   }
```

```
    context() {
        sanddetectedreport0.currentSand.compareTo(self.criticalSandLimit) == 1;
    }
    #reasoning method
    (...)
}
```

Listing D.1: `AnalyseSandContentCriticalSituation`

`ActuateChokePositionChange` had to consider the possibility of receiving several choke adjustment requests at the same time (due to control of sand/water or operator's request), and therefore contained logic enabling the selection of which request to perform. This included some evaluation, so we made a capability called *Actuating choke position change* and two new plans replacing the original:

- `OverrideChokePositionChange`
  If the choke has already been adjusted at the current timetick, this plan decides whether the new request has priority to override the first.
- `NewChokePositonChange`
  If the received request is the first one this timetick, or the request is given by the operator (which has the highest priority) the adjustment can be made with no further evaluation.

To be able to run these plans in a BDI fashion, we also created the plan `SendInternalChokeAdjustmentCommand` which handles `ChokeAdjustmentRequests` and actuates a `ChokeAdjustmentInternalRequest` which is a BDIGoalEvent. Section D.2 describes the use of BDIGoalEvent in more detail.

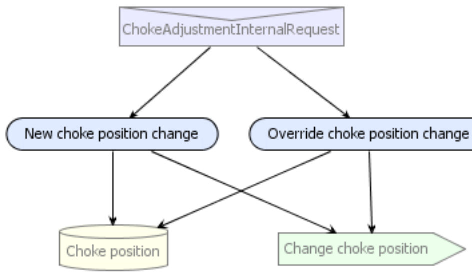

ChokeAdjustmentInternalRequest
New choke position change
Override choke position change
Choke position
Change choke position

Figure D.1: `ActuatingChokePositionChange` in `WellController`

D.1.2 `ProductionOptimiser`

The plan `AnalyseWaterSituation` was split into two new plans which were given a parenting capability *Analysing water situation*. The two new plans are `AnalyseWaterCriticalSituation` and `AnalyseWaterNormalSituation`. The original plan evaluated current situation and performed different subtasks according to whether the situation was critical or not. Now the two new plans each handles one of those situations.

The plan `CalculateOptimalProductionRates` was a complex plan performing many evaluations before deciding on what to do. This plan was therefore made into a capability called *Calculating optimal production rates* which contains the following plans:

- `WaterCritical`
  Handles critical situations by asking the well producing the most water to close the choke. In the original design, critical situations were simply handled by notifying the operator. During implementation we modified the plan so that the agent also closes the well, thereby solving the situation as soon as possible.

- `WaterApproachingCritical`
  Handles the situation of the water being not yet critical but still very high. It chokes several of the wells in order to reduce total water content.
- `WaterHigh`
  Handles the situation where the water is high. Listing D.2 illustrates how the context method of `WaterHigh` checks the amount of water to determine if it should actuate its reasoning method or not. If the context evaluates to `true`, it should find the well producing the most water and ask this to close the choke a bit. Since the situation is not yet critical, it can also ask one of the other wells to open the choke to maintain the total production rate. The reasoning method in Listing D.2 shows how `WaterHigh` performs this task.
- `WaterLow`
  Handles situations where the water content is so low that it does not have to be taken into consideration. In this situation it may ask all the wells to open the choke a bit to optimise the production.

```
public plan WaterHigh extends Plan {
   #uses interface ProductionOptimiser self;
   #uses data aos.jack.util.thread.Semaphore optimise;
   #handles event RankedWellList rankedwelllist0;
   #sends event ProductionAdjustmentRequest productionadjustmentrequest1;
   static boolean relevant(agents.field.plant.RankedWellList ev) {
      return ev.rankedlist!=null;
   }
   context() {//water a bit high
      rankedwelllist0.totalwater.compareTo(new BigDecimal(self.wateroptimiselimit))==1;
   }
   #reasoning method
   body() {
      (...)
      // The well with the largest amount of water in the production
      String worstwell = rankedwelllist0.rankedlist[0].getWellName();
      BigDecimal worstprod = rankedwelllist0.rankedlist[0].getProduction();
      @send(worstwell, productionadjustmentrequest1.adjust(worstprod.multiply(
           adjustmentFactorWorst), worstprod, "water"));

      // The well with the smallest amount of water in the production
      String bestwell = rankedwelllist0.rankedlist[wellCount-1].getWellName();
      BigDecimal bestprod = rankedwelllist0.rankedlist[wellCount-1].getProduction();
      @send(bestwell, productionadjustmentrequest1.adjust(bestprod.multiply(
           adjustmentFactorBest), bestprod, "water"));
      (...)
   }
}
```

Listing D.2: `WaterHigh`

D.1.3 `FieldCoordinator`

Within the capability *Producing alarm* two new plans were added `RemoveSandAlarm` and `RemoveWaterAlarm`. The plans for removing alarms are needed for letting the operator know when the situation has stabilised and is no longer in need of human attention.

D.2 BDIGoalEvent

A regular event can always be handled by several plans, but either all the plans are actuated or the context method must be used to choose the right one. An intelligent agent should according to

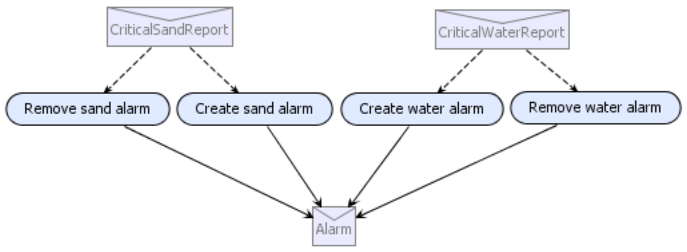


Figure D.2: *Producing alarm*

Section 2.2 be *flexible* and *robust*. This means that it should be able to choose one plan to actuate in order to handle an event. If that plan fails it should be able to actuate others until either one of its plans succeeds or it has no more plans left to try out. This is not possible with regular events, but JACK has an event called BDIGoalEvent that may be used to achieve this kind of behaviour.

The behaviour described is closely related to the behaviour of a BDI agent, as described in Section 2.3. When an agent receives a BDIGoalEvent, the event becomes a part of the agent's desires. When it decides to try to achieve the desire represented by the event, it becomes the agent's *intention*. The agent now enters the loop of choosing a plan, actuating it, and, in case of failure, choose another one until success is achieved.

In JACK, the plans will be actuated in the same order as they are listed in the agent or capability in which they are defined.

D.2.1 Example 1: `CalculatingOptimalProductionRates`

In Section D.1.2 we described the new capability in `WellController`, *Calculating optimal production rates*. The event triggering these plans, `RankedWellList`, is a BDIGoalEvent. Listing D.3 show how `RankedWellList` is defined.

```
public event RankedWellList extends BDIGoalEvent {
   protected WaterWellRelation[] rankedlist;
   protected BigDecimal totalwater;

   #posted as
   (...)
}
```

Listing D.3: The BDIGoalEvent `RankedWellList`

When a `RankedWellList` is received, the first plan, `WaterCritical` is actuated. If that plan fails, `Water approaching critical` sets in. The plans are triggered in the order they are listed in *Calculating optimal production rates*, as seen in Listing D.4.

```
public capability CalculatingOptimalProductionRates extends Capability {
   #handles external event RankedWellList

   #uses plan WaterCritical;
   #uses plan WaterApproachingCritical;
   #uses plan WaterHigh;
   #uses plan WaterLow;

   #imports data Semaphore optimise();
   #sends event ProductionAdjustmentRequest productionadjustmentrequest1;
}
```

Listing D.4: The capability *Calculating optimal production rates*

D.2.2 Example 2: `ActuatingChokePositionChange`

A BDIGoalEvent can only be actuated and posted internally within an agent. If an external message is to be handled the same way, the message must first be handled by a plan which creates a BDIGoalEvent, giving it the same content as the external message. The BDIGoalEvent will then represent the original message when processed internally, so the plans that actually handles the contents of the original message will use the BDIGoalEvent instead. Figure D.3 is a section of the `WellController`'s internal design, which illustrates this. Instead of the `ChokeAdjustmentRequest` (which is an external message that may be sent by another agent) being handled by *Actuating choke position change* directly, it is first handled by *Send internal choke adjustment command* which creates a `ChokeAdjustmentInternalRequest`. The `ChokeAdjustmentInternalRequest` is a BDIGoalEvent which can then be handled by the plans in *Actuating choke position change*.

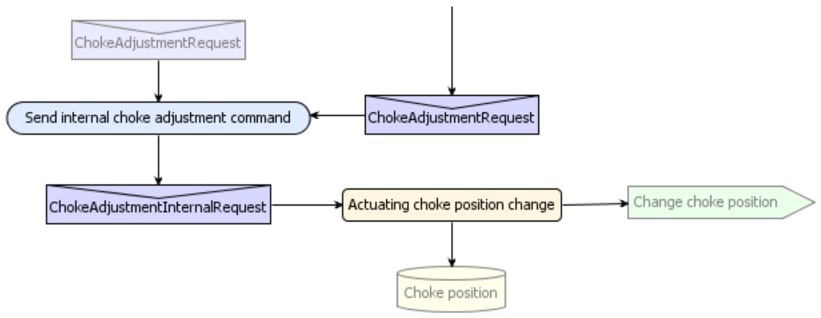

ChokeAdjustmentRequest
Send internal choke adjustment command
ChokeAdjustmentRequest
ChokeAdjustmentInternalRequest
Actuating choke position change
Change choke position
Choke position

Figure D.3: Use of BDIGoalEvent in `WellController`

As shown in Listing D.5, *Actuating choke position change* has two plans, `OverrideChokePositionChange` and `NewChokePositionChange`. They perform the tasks explained in Section D.1.1. If there exists an already saved change, `OverrideChokePositionChange` will find this and therefore succeed, being able to decide whether to carry out the change request or not. `NewChokePositionChange` will in those cases not be actuated since the goal `ChokeAdjusmtentInternalRequest` has been fulfilled. `OverrideChokePositionChange` fails if there is no change saved at the current time already, and in that case, `NewChokePositionChange` is actuated to fulfil the goal.

```
public capability ActuatingChokePositionChange extends Capability {
   #handles external event ChokeAdjustmentInternalRequest;
   #uses plan OverrideChokePositionChange;
   #uses plan ActuateNewChokePositionChange;
   #imports data ChokePosition chokeposition0();
}
```

Listing D.5: The capability *Actuating choke position change*

D.3 Semaphore

A semaphore is *a synchronisation resource which is used to establish mutual exclusion regions of processing in JACK plans and thread* [6]. A plan or a thread may wait for a semaphore before it can actuate its task, and signal on it when it has completed. We use this JACK feature when the agents try to optimise the production with regards to the water content situation.

The optimisation process is composed of several plans contained in three different agents; `ProductionOptimiser`, `WellController` and `WellMonitor`. The `ProductionOptimiser` sends water content requests to all the wells, and it must be able to recognise when it has received replies from all of them. Because of this interaction we found it necessary to make sure only one such process is in progress at any time. The `ProductionOptimiser` has a semaphore called *optimise*. The `SendWaterContentRequest` waits for this semaphore to be released, and then grabs it as shown in Listing D.6. The following plans can then be actuated. When the last plan involved in the process finishes, the semaphore is signalled and thereby released. If `SendWaterContentRequest` is triggered while an earlier optimisation process is still ongoing, it must wait until the semaphore is released before it can continue. Listing D.7 shows how the plan `WaterHigh` within *Calculating optimal production rates* signals the semaphore to release it.

```
public plan SendWaterRatioRequest extends Plan {
   #uses interface ProductionOptimiser self;
   #uses data Semaphore optimise;
   #handles event FieldOptimisationRequest fieldoptimisationrequest0;
   #sends event WaterRatioRequest waterratiorequest2;
   (...)
   #reasoning method
   body() {
      @wait_for(optimise.planWait());
      try{
         for(int i=0; i<self.wellControllers.length; i++){
            String receiver = self.wellControllers[i].getName();
         @send(receiver, waterratiorequest2.request());
      }
   }
}
```

Listing D.6: `SendWaterContentRequest` waiting for the semaphore *optimise*

```
public plan WaterHigh extends Plan {
   (...)
   #reasoning method
   body() {
      (...)
      @send(worstwell, productionadjustmentrequest1.adjust(worstprod.multiply(
           adjustmentFactorWorst), worstprod, "water"));
      (...)
      @send(bestwell, productionadjustmentrequest1.adjust(bestprod.multiply(
           adjustmentFactorBest), bestprod, "water"));

      // Letting go of the semaphore
      try{
         optimise.signal();
      }
   }
}
```

Listing D.7: `WaterHigh` releasing the semaphore *optimise*

APPENDIX E

MESSAGE DESCRIPTORS

[M1] `OperatorChokeAdjustmentRequest`	
Description:	This message is assembled and sent by the `OperatorAssistant` when the operator has requested an adjustment of the choke of a given well.
Sender:	`OperatorAssistant`
Receiver:	`FieldCoordinator`
Information:	A name identifying the well in question, a number denoting the new choke position and the time the request was sent.

Table E.1: [M1] `OperatorChokeAdjustmentRequest`

[M2] `ChokeAdjustmentRequest`	
Description:	When the `FieldCoordinator` receives an OperatorChokeAdjustmentRequest, it determines which `WellController` is responsible for the well in question, and sends a ChokeAdjustmentRequest to it.
Sender:	`FieldCoordinator`
Receiver:	`WellController`
Information:	The time of the original request from the operator, a number denoting the new choke position and who sent the request (the operator).

Table E.2: [M2] `ChokeAdjustmentRequest`

[M3] OptimisationClaim	
Description:	When the `PlantMonitor` discovers a change in the water content, it sends this message to the `ProductionOptimiser` requesting an optimisation of the production.
Sender:	`PlantMonitor`
Receiver:	`ProductionOptimiser`
Information:	The time when the claim is sent, the current water content of the production, as well as a number denoting the change since the previous recorded water content.

Table E.3: [M3] `OptimisationClaim`

[M4] `SandDetectedReport`	
Description:	This message is sent by the `WellMonitor` to its corresponding `WellController` when it detects sand in the well.
Sender:	`WellMonitor`
Receiver:	`WellController`
Information:	The time when the sand was detected, the current production, the current sand content and the previous sand content.

Table E.4: [M4] `SandDetectedReport`

[M5] `CriticalSandReport`	
Description:	When the sand content of a well's production is above a critical limit, the `WellController` will send this message to the `FieldCoordinator` in order to alarm the operator of the control room.
Sender:	`WellController`
Receiver:	`FieldCoordinator`
Information:	The time when the critical situation was discovered and the current sand content.

Table E.5: [M5] `CriticalSandReport`

[M6] `CriticalWaterReport`	
Description:	When the water content in the processing plant is above a given limit, the ProductionOptimiser will send this message to the `FieldCoordinator` to alarm the operator of the control room.
Sender:	`ProductionOptimiser`
Receiver:	`FieldCoordinator`
Information:	The time when the critical situation was discovered, the current water content and the latest change in content.

Table E.6: [M6] `CriticalWaterReport`

[M7] Alarm	
Description:	When the `FieldCoordinator` receives either a CriticalSandReport or CriticalWaterReport it will assemble the information into an appropriate alarm and send this to the `OperatorAssistant`.
Sender:	`FieldCoordinator`
Receiver:	`OperatorAssistant`
Information:	Information about the critical situation; this may include the water content of the plant or sand content of a well.

Table E.7: [M7] `Alarm`

Appendix F

JACK entity overview

The items in blue are the ones added during implementation. The items in red are the ones removed during implementation. ">" means messages and internal events handled by the plan, "<" denotes messages sent or internal events initiated by the plan.

T1 `WellMonitor`

- **P3** `RecordSensorValuesFromWell`
  - **>E1** `WellSensorValues`
  - **<E2** `NewSandSensorValueNotification`
- **P4** `AnalyseSandSensorValues`
  - **>E2** `NewSandSensorValueNotification`
  - **<M4** `SandDetectedReport`
- **P5** `LookUpWaterContent`
  - **>P1M1** `WaterContentRequest`
  - **<P1M2** `WaterContentReply`

T2 `PlantMonitor`

- **P1** `RecordSensorValuesFromPlant`
  - **>E3** `PlantSensorValues`
  - **<E4** `NewWaterSensorValueNotification`
- **P2** `AnalyseWaterSensorValues`
  - **>E4** `NewWaterSensorValueNotifiaction`
  - **<M3** `OptimisationClaim`

T3 `WellController`

- **P6** `AnalyseSandContentSituation`
  - **>M4** `SandDetectedReport`
  - **<M5** `CriticalSandReport`
  - **<P1M3** `ProductionAdjustmentRequest`
- **C4** `AnalysingSandContentSituation`
  - **P6A** `AnalyseSandContentCriticalSituation`
    - **>M4** `SandDetectedReport`
    - **<M5** `CriticalSandReport`
  - **P6B** `AnalyseSandContentNormalSituation`
    - **>M4** `SandDetectedReport`
    - **<M5** `CriticalSandReport`
    - **<P1M3** `ProductionAdjustmentRequest`

- **P7** `CalculateChokePosition`
  - **>P1M3** `ProductionAdjustmentRequest`
  - **<M2** `ChokeAdjustmentRequest`
- **P8A** `SendInternalChokeAdjustmentCommand`
  - **>M2** `ChokeAdjustmentRequest`
  - **<E5** `ChokeAdjustmentInternalRequest`
- **P8** `ActuateChokePositionChange`
  - **>M2** `ChokeAdjustmentRequest`
- **C5** `ActuatingChokePositionChange`
  - **P8B** `OverrideChokePositionChange`
    - **>E5** `ChokeAdjustmentInternalRequest`
  - **P8C** `ActuateNewChokePositionChange`
    - **>E5** `ChokeAdjustmentInternalRequest`
- **C1** `HandlingWaterContentRequest`
  - **P9** `ForwardWaterContentRequest`
    - **<>P1M1** `WaterContentRequest`
  - **P10** `ForwardWaterContentReply`
    - **<>P1M2** `WaterContentReply`

T4 `ProductionOptimiser`

- **P11** `AnalyseWaterSituation`
  - **>M3** `OptimisationClaim`
  - **<M6** `CriticalWaterReport`
  - **<E6** `FieldOptimisationRequest`
- **C6** `AnalysingWaterSituation`
  - **P11A** `AnalyseWaterCriticalSituation`
    - **>M3** `OptimisationClaim`
    - **<M6** `CriticalWaterReport`
  - **P11B** `AnalyseWaterNormalSituation`
    - **>M3** `OptimisationClaim`
    - **<M6** `CriticalWaterReport`
    - **<E6** `FieldOptimisationRequest`
- **P12** `CalculateOptimalProductionRates`
  - **>E7** `RankedWellList`
  - **<P1M3** `ProductionAdjustmentRequest`
- **C7** `CalculatingOptimalProductionRates`
  - **P12A** `WaterCritical`
    - **>E7** `RankedWellList`
    - **<P1M3** `ProductionAdjustmentRequest`
  - **P12B** `WaterApproachingCritical`
    - **>E7** `RankedWellList`
    - **<P1M3** `ProductionAdjustmentRequest`
  - **P12C** `WaterHigh`
    - **>E7** `RankedWellList`
    - **<P1M3** `ProductionAdjustmentRequest`

P12D `WaterLow`
- **>E7** `RankedWellList`
- **<P1M3** `ProductionAdjustmentRequest`

C2 `RankingWells`

P13 `SendWaterContentRequest`
- **>E6** `FieldOptimisationRequest`
- **<P1M1** `WaterContentRequest`

P14 `ReceiveWaterContentReply`
- **>P1M2** `WaterContentReply`
- **<E8** `UnrankedWellList`

P14A `ReceiveDuplicateWaterContentReply`
- **>P1M2** `WaterContentReply`

P15 `RankWells`
- **>E8** `UnrankedWellList`
- **<E7** `RankedWellList`

T5 `FieldCoordinator`

P16 `ForwardChokeAdjustmentRequest`
- **>M1** `OperatorChokeAdjustmentRequest`
- **<M2** `ChokeAdjustmentRequest`

C3 `ProducingAlarm`

P17 `CreateSandAlarm`
- **>M5** `CriticalSandReport`
- **<M7** `Alarm`

P17A `RemoveSandAlarm`
- **>M5** `CriticalSandReport`
- **<M7** `Alarm`

P18 `CreateWaterAlarm`
- **>M6** `CriticalWaterReport`
- **<M7** `Alarm`

P18A `RemoveWaterAlarm`
- **>M6** `CriticalWaterReport`
- **<M7** `Alarm`

T6 `OperatorAssistant`

P19 `AcceptOperatorsRequest`
- **>E9** `OperatorsChokeAdjustment`
- **<M1** `OperatorChokeAdjustmentRequest`

P20 `PresentAlarm`
- **>M7** `Alarm`

Appendix G

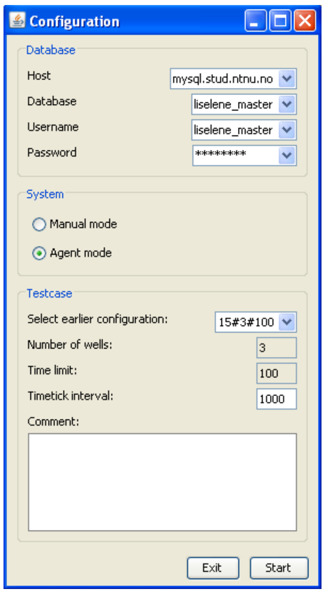
User manual

The simulator is a Java program which simulates the oil production of a very simplified oil field. It was developed in order to provide an environment in which to test our agent system and is therefore equipped with features which enables logging of all events and actions.

G.1 Getting started

The simulator and agent system are both included in the executable `simulator.jar` file. In order to run this file, Java 6 must be installed on your computer and you need access to a MySQL database with the tables found in the included zip-file under `sql/liselene_master.sql`. The simulator has only been tested on the operating system Windows XP. When running, the simulator will create two folders, config and report, in the same location as the jar-file itself. For each run, new folders will be created inside these folders.

G.2 Configuration



Configuration

Database

Host	mysql.stud.ntnu.no
Database	liselene_master
Username	liselene_master
Password	********

System

Manual mode [✓ Agent mode]

Testcase

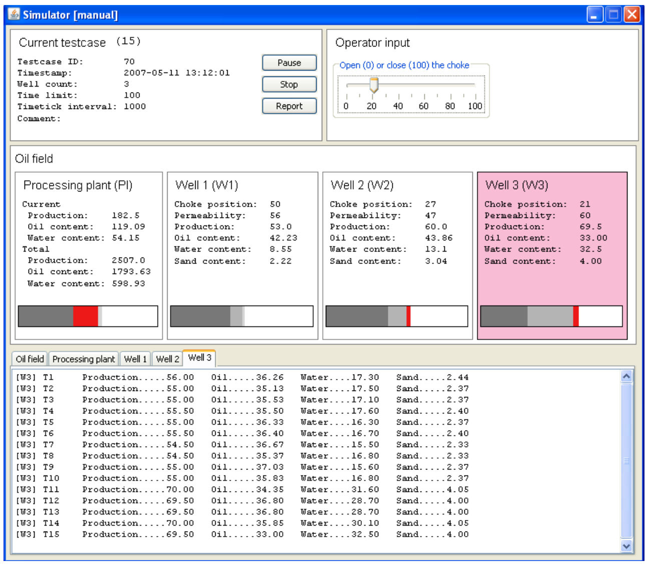
Select earlier configuration:	15#3#100
Number of wells:	3
Time limit:	100
Timetick interval:	1000
Comment:	

Figure G.1: Configuring the simulator

The first screen of the application is shown in Figure G.1 and enables configuration of the testcase to be run. The upper section is for configuring the connection details for the database. You then choose whether the system should run in manual or agent mode. In manual mode, all adjustments of the oil wells must be performed by the user. In agent mode, the agent system will do what it can in order to maximise the oil production of the field. The lower section is for setting the configurable details of the testcase. The simulator can handle up to four wells in the field. The time limit denotes how long the testcase should run and must be an integer between 10 and 500. The timetick interval is the amount of milliseconds between each timetick and should be set to an integer larger than 1000. At the bottom of the screen is a text area for comments related to the current testcase. The first option in the testcase section enables the selection of earlier configurations. The names relates to the names of the respective configuration folders and are given on the following format `<testcaseId>#<wellCount>#<timelimit>`.

G.3 Manual mode

A screenshot of the simulator running in manual mode is shown in Figure G.2. The following paragraphs describe each of the components of the user interface.



```
Current testcase  (15)
Testcase ID:        70
Timestamp:          2007-05-11 13:12:01
Well count:         3
Time limit:         100
Timetick interval:  1000
Comment:
```

```
Processing plant (PI)
Current
 Production:     182.5
 Oil content:    119.09
 Water content:  54.15
Total
 Production:     2507.0
 Oil content:    1793.63
 Water content:  598.93
```

```
Well 1 (W1)
Choke position:   50
Permeability:     56
Production:       53.0
Oil content:      42.23
Water content:    8.55
Sand content:     2.22
```

```
Well 2 (W2)
Choke position:   27
Permeability:     47
Production:       60.0
Oil content:      43.86
Water content:    13.1
Sand content:     3.04
```

```
Well 3 (W3)
Choke position:   21
Permeability:     60
Production:       69.5
Oil content:      33.00
Water content:    32.5
Sand content:     4.00
```

```
[W3] T1     Production.....56.00   Oil.....36.26   Water....17.30   Sand.....2.44
[W3] T2     Production.....55.00   Oil.....35.13   Water....17.50   Sand.....2.37
[W3] T3     Production.....55.00   Oil.....35.53   Water....17.10   Sand.....2.37
[W3] T4     Production.....55.50   Oil.....35.50   Water....17.60   Sand.....2.40
[W3] T5     Production.....55.00   Oil.....36.33   Water....16.30   Sand.....2.37
[W3] T6     Production.....55.50   Oil.....36.40   Water....16.70   Sand.....2.40
[W3] T7     Production.....54.50   Oil.....36.67   Water....15.50   Sand.....2.33
[W3] T8     Production.....54.50   Oil.....35.37   Water....16.80   Sand.....2.33
[W3] T9     Production.....55.00   Oil.....37.03   Water....15.60   Sand.....2.37
[W3] T10    Production.....55.00   Oil.....35.83   Water....16.80   Sand.....2.37
[W3] T11    Production.....70.00   Oil.....34.35   Water....31.60   Sand.....4.05
[W3] T12    Production.....69.50   Oil.....36.80   Water....28.70   Sand.....4.00
[W3] T13    Production.....69.50   Oil.....36.80   Water....28.70   Sand.....4.00
[W3] T14    Production.....70.00   Oil.....35.85   Water....30.10   Sand.....4.05
[W3] T15    Production.....69.50   Oil.....33.00   Water....32.50   Sand.....4.00
```

Figure G.2: Simulator in manual mode

Current testcase Displays the configuration details of the current testcase. The time will run until it reaches the timelimit, but may be paused using the button labeled *Pause*. The *Stop* button will finalize the testcase and does not allow it to be restarted. The *Report* button can be used at any time and will lead to the generation of reports in the `reports/testcase<testcaseID>` folder. One report for each component (wells and plant) of the field is generated and the files are saved with names on the format `<testcaseID>#<componentName>#<currentTime>.csv`.

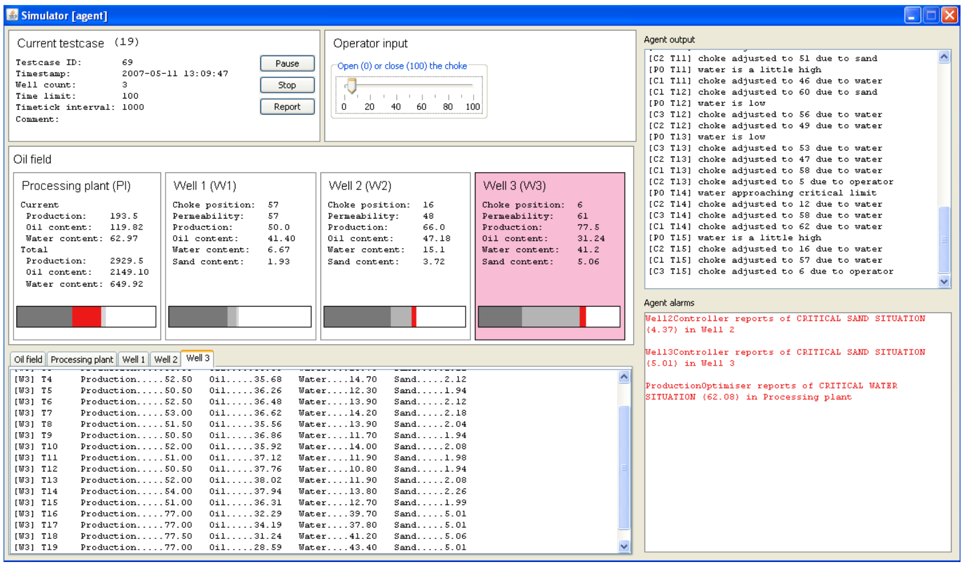
Operator input The operator controls the choke of each of the wells. In order to adjust a choke, the respective well must be selected (explained in the next section) and the slider representing the choke must be moved to the desired position. A choke position of 0 means that it is completely open, 100 means that it is completely closed.

Oil field This frame displays the current state of each of the components of the field. A field consists of a processing plant and a set of wells. For each timetick, all the wells produce variable amounts of oil, water and sand. The production from all wells are received for processing by the processing plant. The components are all displayed with relevant information concerning their current production and a bar shows the relative amounts of oil, water and sand produced or received by each component. Dangerous levels of sand or water in the production are marked in red. The component (and the field itself) may be selected by clicking on it with the left mouse button.

System output This tabbed pane enables the user to view the development of each field component over time. Selecting a component through clicking on a tab will also select it in the oil field pane, and vice versa. One line in the output shows the code of the component in question, the time of sensor reading, and the values for total production, oil, water, and sand.

G.4 Agent mode

A screenshot of the simulator running in agent mode is shown in Figure G.3. The user interface is the same as the manual mode user interface, with the addition of text panes for agent output. The upper pane is used for general output from the agents. The lower pane is used for displaying alarms from the agents to the operator of the control room. The operator may still use the slider which represents the choke in the user interface. The difference from manual mode choke adjustments is that moving the slider results in a request being sent to the agents, rather than the operator being able to control the choke directly.



```
Testcase ID:        69
Timestamp:          2007-05-11 13:09:47
Well count:         3
Time limit:         100
Timetick interval:  1000
Comment:
```

```
Processing plant (PI)
Current
 Production:     193.5
 Oil content:    119.82
 Water content:  62.97
Total
 Production:     2929.5
 Oil content:    2149.10
 Water content:  649.92
```

```
Well 1 (W1)
Choke position:   57
Permeability:     57
Production:       50.0
Oil content:      41.40
Water content:    6.67
Sand content:     1.93
```

```
Well 2 (W2)
Choke position:   16
Permeability:     48
Production:       66.0
Oil content:      47.18
Water content:    15.1
Sand content:     3.72
```

```
Well 3 (W3)
Choke position:   6
Permeability:     61
Production:       77.5
Oil content:      31.24
Water content:    41.2
Sand content:     5.06
```

```
[W3] T4     Production.....52.50   Oil.....35.68   Water....14.70   Sand.....2.12
[W3] T5     Production.....50.50   Oil.....36.26   Water....12.30   Sand.....1.94
[W3] T6     Production.....52.50   Oil.....36.48   Water....13.90   Sand.....2.12
[W3] T7     Production.....53.00   Oil.....36.62   Water....14.20   Sand.....2.18
[W3] T8     Production.....51.50   Oil.....35.56   Water....13.90   Sand.....2.04
[W3] T9     Production.....50.50   Oil.....36.86   Water....11.70   Sand.....1.94
[W3] T10    Production.....52.00   Oil.....35.92   Water....14.00   Sand.....2.08
[W3] T11    Production.....51.00   Oil.....37.12   Water....11.90   Sand.....1.98
[W3] T12    Production.....50.50   Oil.....37.76   Water....10.80   Sand.....1.94
[W3] T13    Production.....52.00   Oil.....38.02   Water....11.90   Sand.....2.08
[W3] T14    Production.....54.00   Oil.....37.94   Water....13.80   Sand.....2.26
[W3] T15    Production.....51.00   Oil.....36.31   Water....12.70   Sand.....1.99
[W3] T16    Production.....77.00   Oil.....32.29   Water....39.70   Sand.....5.01
[W3] T17    Production.....77.00   Oil.....34.19   Water....37.80   Sand.....5.01
[W3] T18    Production.....77.50   Oil.....31.24   Water....41.20   Sand.....5.06
[W3] T19    Production.....77.00   Oil.....28.59   Water....43.40   Sand.....5.01
```

```
[C2 T11] choke adjusted to 51 due to sand
[P0 T11] water is a little high
[C1 T11] choke adjusted to 46 due to water
[C1 T12] choke adjusted to 60 due to sand
[P0 T12] water is low
[C3 T12] choke adjusted to 56 due to water
[C2 T12] choke adjusted to 49 due to water
[P0 T13] water is low
[C3 T13] choke adjusted to 53 due to water
[C2 T13] choke adjusted to 47 due to water
[C1 T13] choke adjusted to 58 due to water
[C2 T13] choke adjusted to 5 due to operator
[P0 T14] water approaching critical limit
[C2 T14] choke adjusted to 12 due to water
[C3 T14] choke adjusted to 58 due to water
[C1 T14] choke adjusted to 62 due to water
[P0 T15] water is a little high
[C2 T15] choke adjusted to 16 due to water
[C1 T15] choke adjusted to 57 due to water
[C3 T15] choke adjusted to 6 due to operator
```

```
Well2Controller reports of CRITICAL SAND SITUATION
(4.37) in Well 2

Well3Controller reports of CRITICAL SAND SITUATION
(5.01) in Well 3

ProductionOptimiser reports of CRITICAL WATER
SITUATION (62.08) in Processing plant
```

Figure G.3: Simulator in agent mode

G.5 Further development

We have developed the simulator and agent system in Eclipse SDK version 3.3.0. Compiling and running the agent system has been done using the JACK Intelligent Agents Compiler, Runtime Environment and BDI Agent Model.

The Eclipse project can be found in the included zip-file and its content is briefly described below;

`source/agents`
Various Java classes used by the agent system

`source/agents.*`
Java classes generated by the JACK compiler

`source/simulator.*`
The source code of the simulator

`config`
Generated datasets for environment variables

`jack`
JACK agent code

`lib`
Required libraries (JACK and MySQL)

`report`
Generated testrun reports

`sql`
Definitions of the required database tables

Appendix H

Testruns

The graphs showing the development of the production over time for all field components are displayed here. Figure H.1 to H.4 shows the graphs for the testrun without input. Figure H.5 to H.8 shows the graphs for the operator testrun. Figure H.9 to H.12 shows the graphs for the agent testrun.

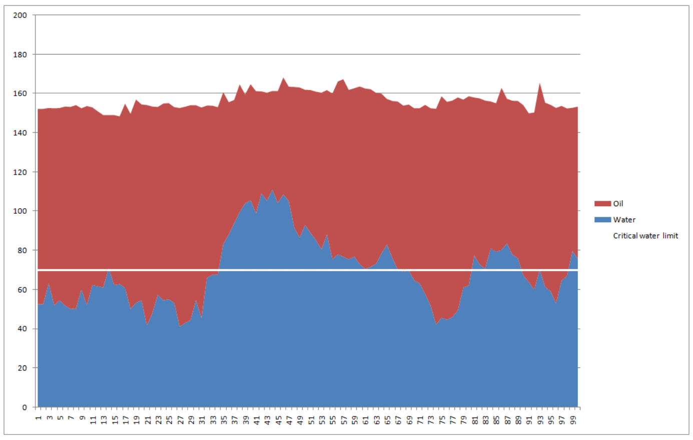


Figure H.1: Testrun with no input - plant

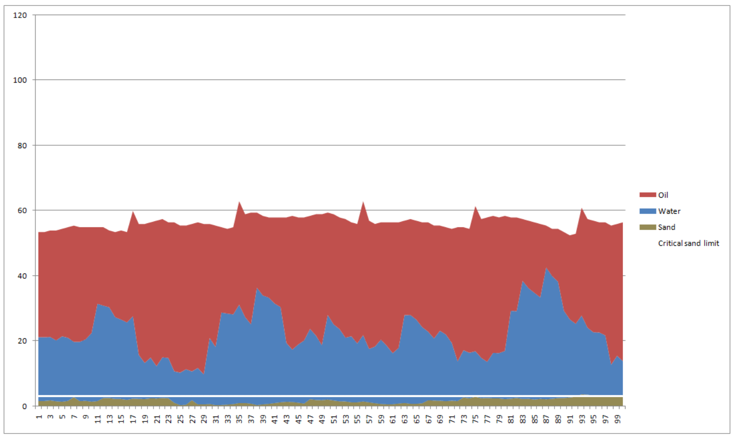


Figure H.2: Testrun with no input - well 1

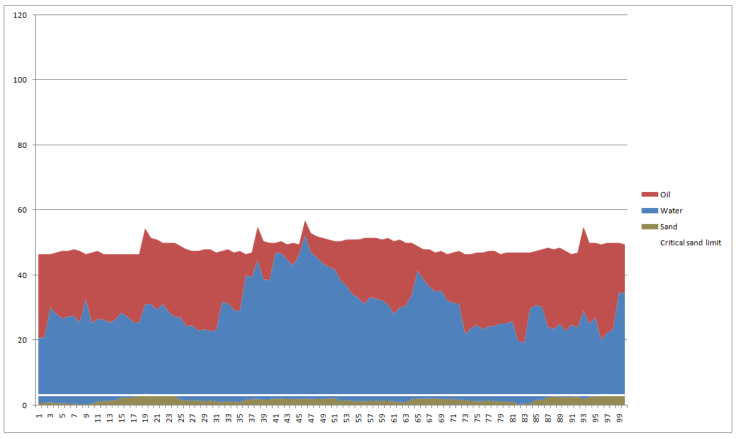


Figure H.3: Testrun with no input - well 2

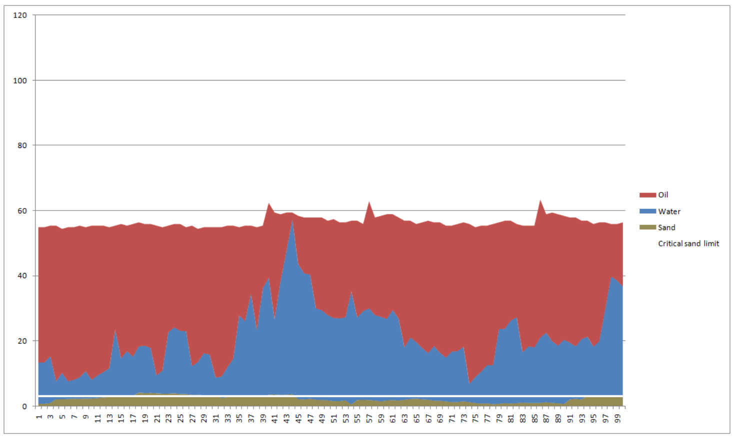


Figure H.4: Testrun with no input - well 3

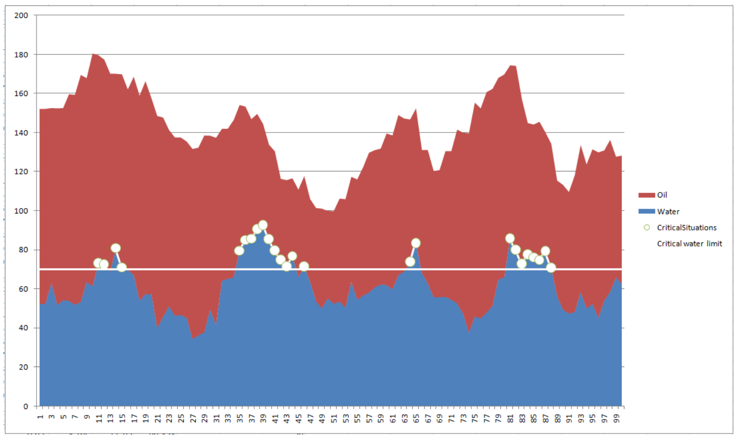


Figure H.5: Operator testrun - plant

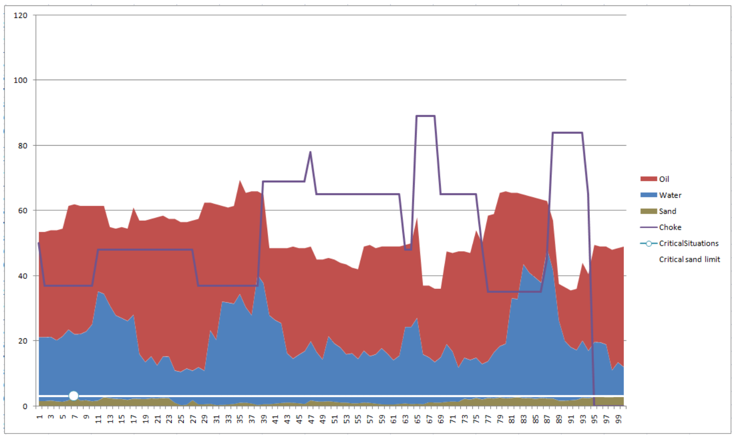


Figure H.6: Operator testrun - well 1

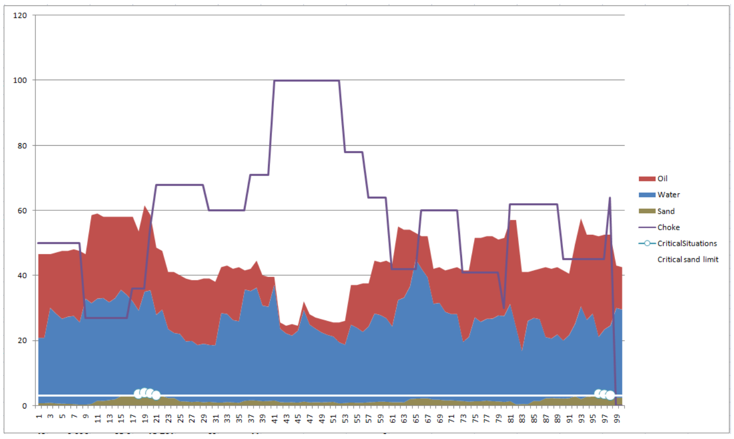


Figure H.7: Operator testrun - well 2

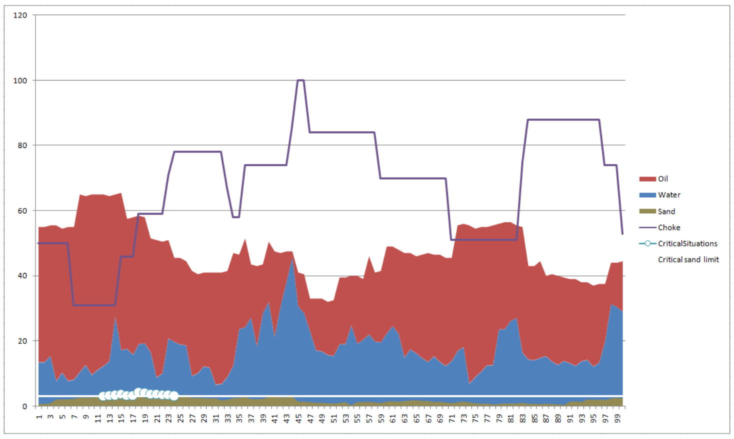


Figure H.8: Operator testrun - well 3

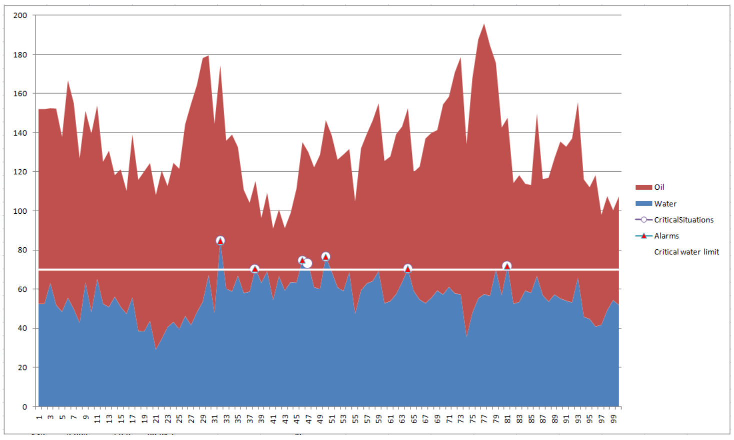


Figure H.9: Agent testrun - plant

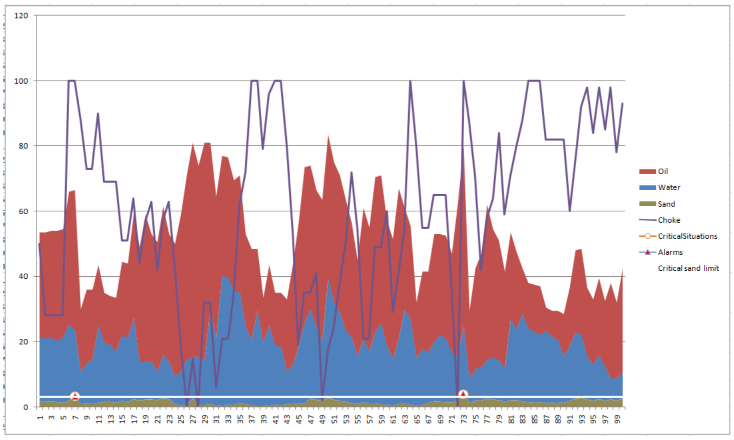


Figure H.10: Agent testrun - well 1

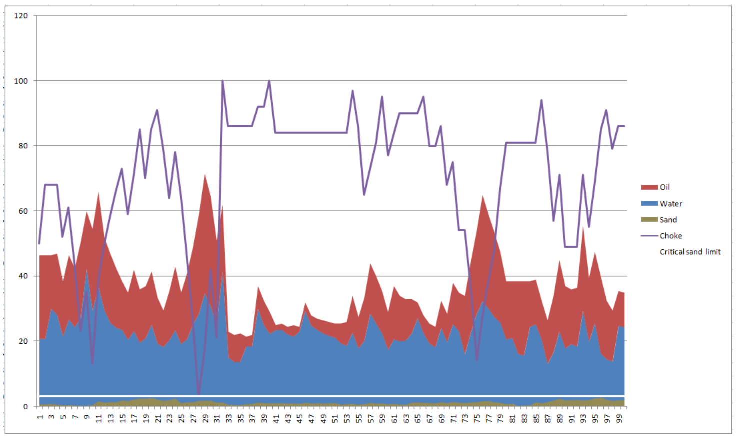


Figure H.11: Agent testrun - well 2

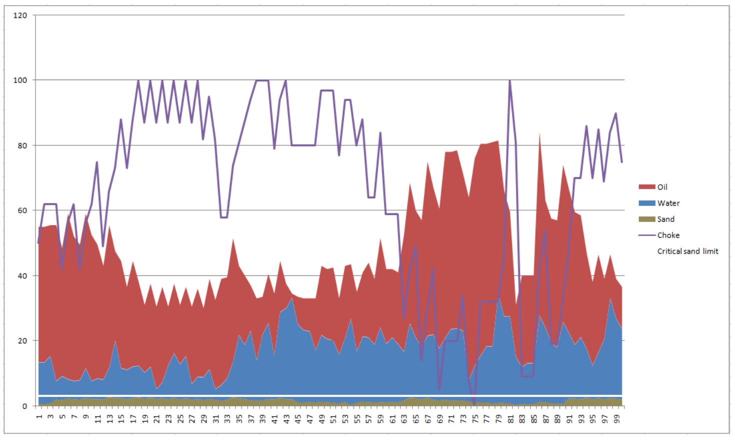


Figure H.12: Agent testrun - well 3

Appendix I

Acronyms

AOS Agent Oriented Software

AUML Agent UML

BDI Belief-desire-intention

CBR Case-based reasoning

CVS Concurrent versions system

dMARS The distributed Multi-Agent Reasoning System

GUI Graphical user interface

GQM Goal/Question/Metrics

IDE Integrated development environment

IDI Department of Computer and Information Science

IFM Integrated Field Management

JDE JACK Development Environment

JVM Java Virtual Machine

NTNU Norwegian University of Science and Technology

PDT Prometheus Design Tool

PRS Procedural Reasoning System

POC proof-of-concept

SDK Software Development Kit

BIBLIOGRAPHY

[1] Eduardo Alonso, Mark D´Inverno, Daniel Kudenko, Michael Luck, and Jason Noble. Learning in multi-agent systems. *The Knowledge Engineering Review*, 16:3, 2001. 2.2

[2] Barry Burd. *Eclipse For Dummies*. Wiley, 2005. 3.4

[3] Charles F. Conaway. *The Petroleum Industry - A Nontechnical Guide*. PennWell Publishing Company, 1999. 4.1, 4.1.1, 1, 4.1.2, 2

[4] The Eclipse Foundation. Eclipse. URL `http://www.eclipse.org`. [online; accessed 02.05.2007]. 3.4

[5] Shaw Green, Leon Hurst, Brenda Nangle, Pádraig Cunningham, Fergal Somers, and Richard Evans. Software agents: A review, May 1997. URL `https://www.cs.tcd.ie/research_groups/aig/iag/toplevel2.html`. 2.3

[6] The Agent Oriented Software Group. Jack intelligent agents agent manual. http://www.agent-software.com/shared/demosNdocs/Agent_Manual.pdf, 2005. 3.3, D.1, D.3

[7] The Agent Oriented Software Group. Jack intelligent agents agent practicals. http://www.agent-software.com/shared/demosNdocs/practicals/jack_jde/practicals.pdf, 2005. 3.3

[8] The Agent Oriented Software Group. Intelligent software for the new millennium. http://www.agent-software.com/shared/resources/brochures/JACKBrochure-AU-UK.pdf, . 3.3

[9] The Agent Oriented Software Group. The agent oriented software group, . URL `http://www.agent-software.com`. 1.3, 3.3

[10] Lene Hallen and Lise Engmo. Using software agents to reduce information overload in modern oil and gas production. Technical report, 2006. 1.1, 2.2

[11] Brian Henderson-Sellers and Paolo Giorgini. *Agent-Oriented Methodologies*. Idea Group Publishing, 2005. 3.1

[12] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, April 2001. 2.1

[13] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2001. 2.5

[14] Raymond Paul Lees. Increase oil production and reduce chemical usage through separator level measurement by density profiling. In *ISA Emerging Technologies Conference*, 2002. 4.1.2

[15] Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems - A practical guide*. John Wiley & Sons, Ltd, 2004. 3.1

[16] A. S. Rao and M. P. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, 1995. 2.3

[17] Schlumberger. Oilfield glossary. URL `http://www.glossary.oilfield.slb.com`. [online; accessed 10.04.2007]. (document), 4.3

[18] Sandip Sen and Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 1999. 2.2

[19] Statoil. Fakta om gullfaks. URL `http://www.statoilnorge.no/STATOILCOM/SVG00990.nsf?opendatabase&lang=no&artid=C18B58F274DB72BAC1256FE00029C519`. [online; accessed 20.05.2007]. 4.1.4

[20] Katia P. Sycara. Multiagent systems. *AI Magazine*, 1998. 2.4, 2.5

[21] John Thangarajah, Lin Padgham, and Michael Winikoff. Prometheus design tool. In *Proceedings of the fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 127–128, New York, NY, USA, 2005. ACM Press. 3.2, 3.2

[22] Wikipedia. Wikipedia, 2007. URL `http://en.wikipedia.org/`. [online; accessed 03.03.2007]. (document), 4.1, 4.2

[23] Michael Winikoff. Towards making agent uml practical: A textual notation and a tool. In *Integration of Software Engineering and Agent Technology*, September 2005. B.2

[24] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000. 5.1, 5.2, 5.3.1, 5.3.2, 5.3.4, 5.3.5

[25] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd, 2002. 2.2, 2.3, 2.4