NTNU
Norwegian University of
Science and Technology

A Comparison between JACK Intelligent Agents and JACK Teams Applied in Teamwork

Øystein Spillum

Master of Science in Computer Science

Submission date:	November 2008
Supervisor:	Harald Rønneberg, IDI
Co-supervisor:	Einar Landre, StatoilHydro
	Jørn Ølmheim, StatoilHydro

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

Problem Description

Is it easier to develop teamwork in JACK Teams than in JACK Agents, when building a decision-support system? Is it possible to develop a similar system in JACK Intelligent Agents and JACK Teams? Will JACK Teams be a more feasible platform than JACK Intelligent Agent, when developing teamwork?
The reference problem and system design made to test the two modelling paradigms, shall be based on and continue the work made in the author's depth study.

Assignment given: 15. June 2008
Supervisor: Harald Rønneberg, IDI

Abstract

Modern technology enables the oil industry to develop smarter solutions that improve their work processes. Cheap network bandwidth results in improved communication between offshore and onshore. The availability of sensor data is increasing. Toghether, this will enable a more optimal decision making in oil production. Three challenges have to be dealt with to optain this: information overload of signals generated by equipment, shared situation awareness between technical experts, and mutually-agreed timeframe for action. These challenges are addressed in *human-centric systems*, which have an extensive use of *teamwork*. Research in teamwork focuses on the human-machine interactions, and getting humans included in teamwork processes. This will cause increased situation awareness capability for humans when dealing with unknown or hostile environments. The environment of oil production has a similar characterization. Teamwork can therefore be a possible improvement in the decision-making regarding oil production. The construction of teamwork is examined in this thesis through the two modeling paradigms contained in the JACK framework. The two modeling paradigms are *JACK Intelligent Agents* and *JACK Teams*.

This report investigates JACK Intelligent Agents and JACK Teams, and makes a comparison between the two. The main object was to find indications that point out which modeling paradigm that results in least development effort, and which one that is creating the most feasible platform regarding teamwork construction. The application domain is *decision-support systems used in oil production*. The aspects evaluated are *development effort*, *degree of coupling*, *encapsulation of functionality*, *abstraction level*, *delegation of autonomy*, and *scalability*. The solutions developed in the comparison had static team formations that included few teammembers. This caused less development effort by using JACK Intelligent Agents, and was the main reason why it was considerate to be the preferred modeling paradigm in this case. This was partly experienced because reasoning based on the actual team membership was not used in the JACK Teams version. The use of roles was used instead, causing more JACK entities where it could have been avoided. Dynamic team formations during runtime were not needed due to the reference problem introduced. Maintaince during runtime, for instance introducing new subteams and changing the role structure was not looked into. Introducing teams in large scale was not performed. These four factors could have caused a different result. The question is if JACK Teams shows its potential through the oil production system designed in this report.

Preface

This project report documents my work in the graduate level course *TDT4900 Program and Information Systems, Master Thesis*. The course is a part of the Master program at Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU)

The subject of this report has been carried out in collaboration with StatoilHydro. The supervisors of this project have addressed the subject as a research area of their company, and have contributed with knowledge and feedback during the experiment conducted and the writing process.

I would like to thank my supervisors Harald Rønneberg, Einar Landre, and Jørn Ølmheim for introducing an interesting field of research, and for sharing their knowledge and support during the project.

Trondheim, 15. November 2008

____

Øystein Spillum

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This introduction chapter deals with the main objectives and context of the report. The first section explains the motivation for conduction the work of this report. Based on this, the problem definition addresses the area of research. An extraction of the important goals based on the problem definition is shown later in the chapter, followed by the approach to these goals. At the end, a short description for each of the following chapters can be found.

1.1 Motivation

Modern technology enables the oil industry to develop smarter solutions that improve their work processes. Cheap network bandwidth results in improved communication between offshore and onshore. The availability of sensor data is increasing. Toghether, this will enable a more optimal decision making (1). Three important aspects of this development are

- *Information overload* of signals generated by equipment. The control room and human operators cannot focus on all signals, when each well and processing equipment generates several signals at any time.
- *Shared situation awareness* between technical experts. The different experts can have different perspectives of priority and criticality in different situations.
- *Mutually-agreed timeframe* for action. A rapid response time ensure optimization of oil production.

The three aspects listed require support from information systems to be able to make the appropriate abstraction, and to have tools for automation and decision support. Some academic studies from the Norwegian University of Science and Technology (NTNU) on initiative from StaoilHydro have documented research related to these aspects. StatoilHydro also shows experiences made by earlier developments of relevant multi-agents in the report written by the authors Ølmheim, Landre, & Quale (2). This report also describes the basis of future systems, and research areas related to it. Their work leads towards human centric systems.

The depth study by Spillum (3) was motivated by the work towards human centric systems, and forms the fundament of the work in this report. The depth study mentioned was written by the same author as this report, and will therefore have a natural continuation in this work. The reference problem[1] and system design that are used in this report is based on the depth study. By inheritating these two elements, so will the limitations they create. The depth study considers a stronger use of teamwork in such a production system. The team concept is examined in this report by using both modelling paradigms, JACK Intelligent Agents and JACK Teams. A comparison between the two will explore the applicability and suitability for teams of agent-instances (in JACK Agents) and teams-

[1] Specified problem found in the application domain: *decision-support systems used in oil production.* The reference problem is described in Chapter 5.

instances (in JACK Teams) in the development of a decision-support system used in oil production. This comparison is the main goal for this report and the motivation for StatoilHydro.

1.2 Problem definition

Based on the motivation presented in the last section, the problem definition was formulated in collaboration with the supervisors:

"Is it easier to develop teamwork in JACK Teams than in JACK Agents, when building a decision-support system? Is it possible to develop a similar system in JACK Intelligent Agents and JACK Teams? Will JACK Teams be a more feasible platform than JACK Intelligent Agent, when developing teamwork?

The reference problem and system design made to test the two modelling paradigms, shall be based on and continue the work made in the author's depth study (3)***."***

1.3 Project goal

Extracted from the problem definition, the project goals define what to look for as important results in the proposed solutions. The main goals are to:

1. Compare the easiness and results of developing a production system in the two modelling paradigms.
2. See how to construct an oil production system that makes it easy to maintain and replace software-components.
3. See how to produce oil volumes according to the long term production targets (a production plan) given the system within the equipment's capacities.
4. See how to handle unexpected situations caused by the complex and dynamic oil production environment.
5. See how to construct a scalable solution (looking at the aspect of system development, and not necessarily performance), with the ability to contain hundreds of wells and other physical components used in oil production (real life realistic).

1.4 Approach

This section describes how to approach the project goals from Section 1.3. Some of the important approaches to fulfil the goals are to:

- Create a production system with the same architecture, rules, and capabilities in both modelling paradigms.
- Predict the future production in order to create a long term optimization plan that the system can follow, in order to reach the production targets of a given period of time.
- Make proactive decisions (about adjusting well chokes) when the system has knowledge about how the environment will react to its decisions, due to forecasting. This makes it easier to optimize in long term, knowing when to make adjustments and what the effects will be.
- Conduct compensating actions if a production well stops producing or decreases production because of technical failure or unpredicted changes reservoir- and/or well condition. Compensating actions will be if one well takes over production for another well, because they are assumed to have a better oil/waste production rate at the moment. Although this might not be the best wells to choose over a longer period, as changing well chokes rapidly

does not make immediate changes in the production. Fluids and other substances need some time to react to the changed choke settings. If the best oil/waste production rates were known in advance, the well settings could have been changed before the oil/waste production rates changes (proactive behaviour), and not afterwords like this reactive behaviour causes.

- Create cooperation between autonomous units (agent-instances, teams-instances and human operators), with different levels of authority and autonomy.

1.5 Use of terms

The team-construct will use the notation team-construct or team-instance, if it is not clear if the term is used in context of JACK Teams or as a team in context of teamwork. In JACK Agents a team is an agent being a team-commander, with teammembers being agents connected to it. In JACK Teams a team-instance forms a team, with teammembers being other team-instances connected to it.

The modelling paradigm JACK Intelligent Agents will be abbrivated to JACK Agents.

JACK Agents and JACK Teams are referred to as both modelling paradigms and programming languages in this report.

1.6 Project context

This project is conducted in collaboration with StatoilHydro, as a part of the graduate level course *TDT4900 Program and Information Systems, Master Thesis*. StatoilHydro operates in about 40 countries and has about 31,000 employees. The company is one of the largest operators in the Norwegian oil- and gas production, and the international production is increasing (4). In order to be competitive and flexible according to the market and oil production, a more optimal production is wanted.

1.7 Report outline

The report is divided into following chapters:

Chapter 2, Towards human centric systems This chapter introduces the context of the system design. Human and machine are mutual dependent in complex and changing environments, and will lead towards a human centric system. Teamwork and delegation of autonomy are two important elements in this trend, and will be described in depth in this chapter.

Chapter 3, Framework and Tools This chapter introduces the development tools and the modelling paradigms JACK Agents and JACK Teams made use of to design and implement the system used in the experiment conducted in this report.

Chapter 4, Approach This chapter introduces the approach that will test JACK Agents and JACK Teams regarding their applicability and suitability to implement teamwork. The approach is described as an experiment to be evaluated quantitative and qualitative. The experiment is designed in this chapter.

Chapter 5, Defining a reference problem This chapter introduces the type of system that JACK Agents and JACK Teams shall be tested against. The reference problem creates the context to test teamwork within, and will therefore create both possibilities and limitations on how teamwork can be tested.

Chapter 6, System design This chapter constructs a system design presenting the different architectual layers, control processes, and “steps of action”-scenarios that address the reference problem described in Chapter 5. The system design presented in this chapter is used as a basis in the the JACK Agent version and JACK Teams version, and to create similar and comparable versions of the system.

Chapter 7, JACK Agents solution Based on the system design in Chapter 6, this chapter describes the specialized design and implementation of the JACK Agents version of the system. The design shows the system structure of all the agents that will be instanciated, and how the different kinds of agent-instances interact with eachother to enable teamwork.

Chapter 8, JACK Teams solution Based on the system design in Chapter 6, this chapter describes the specialized design and implementation of the JACK Teams version of the system. The design shows the system structure of all the agents-instances and teams-instances that will be instanciated, and how the different kinds of agent-instances and team-instances interact with eachother to enable teamwork.

Chapter 9, Quantitative results This chapter contains results from the quantitative part of the experiment conducted in this report. The JACK Agents solution and the JACK Teams solution are evalutated in a quantitative manner in accordance to the differenct quantitative evalutation aspects defined in Chapter 4.

Chapter 10, Qualitative results This chapter describes advantages and disadvantages created by JACK Agents and JACK Teams when constructing teamwork. The evaluation is given in accordance to the differenct qualitative evalutation aspects defined in Chapter 4.

Chapter 11, Summary of work This chapter contains a summary of experiences made during the use of the new programming techniques JACK Teams introduced compared to JACK Agents.

Chapter 12, Conclusion This chapter draws the conclusion about which modelling paradigm that is most applicable and suitable in teamwork construction.

Chapter 13, Future work This chapter deals with areas in the JACK Teams modelling paradigm that should be further investigated, and suggested improvements of the proposed system that is used as relevant test-object.

Part I

State of the art

Chapter 2

Towards human-centric systems

This chapter introduces the trend towards human-centric systems. Human-centric systems are important in decision-support needed in complex and changing environments like oil production. Humans and machines are mutual dependent in such kind of environments. This chapter describes how the decision-support development will move towards human-centric systems, and creates new software demands. The focus of the report is how teamwork addresses these new demands. Teamwork is described in depth in Section 2.5.

2.1 Human and machine as mutually dependent

Decision-support system requires two actors, the human operator and the machine. The machine system is in this report reffered to as a multi-agent system. The term agent[2] does not have a universal definition in the literature. A common defition is made by Wooldridge (3):

"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. An intelligent agent is in addition reactive, proactive, and social."

Agent-oriented development can be considered as the successor of objectoriented development when applied in artificial intelligence problem domains (5). More information about agents and their application areas can be found in the author's depth study (3).

Challenges in *multi-agent systems*[3] are (3) that they have no existing *global system control*, data is *decentralized*, and that the computation is *asynchronous*. *Teamwork* is taken into consideration to address these challenges. By definition, teamwork is a group of agents that works towards a common goal. Such work requires the system to consider the challenges mentioned. Another element that can be problematic for a multi-agent system is the interface against human operators. The human operator has to build a relationship of *trust* to the system, since most human operators are not comfortable giving away their authority. Teamwork will include human operators in the team formations and in the team processes, and possibly cause an increase of trust.

The agent architecture considered in this report through the JACK framework is called *Belief-Desire-Intention* (BDI architecture). This architecture makes use of a human-like intelligence- and decision making behavior (6). This behaviour makes the agents a possible substitution for humans in teamwork. The BDI architecture contains both *reactive* and *deliberative* properties, and models the mental states of an agent. These mental states are *belief*, *desire*, and *intentions*. The different states will cause behavior on an abstraction level closer to human intelligence. This kind of behavior is realized through an event-driven execution model; wich enables both reactive and proactive behavior (7). Belief is defined as the agent's view about the environment. The belief will change over

[2] The term agent refers in this chapter to an agent-instance in JACK Agents and as a team-instance in JACK Teams. A team-instance is an extension of an agent-instance, in the JACK Teams modelling paradigm.
[3] Multi-agent systems are composed of multiple interacting agents who may be distributed (24).

time in order to be consistent with the environment. The desire is goals the agent wants to achieve. Desire has to be consistent with the agent's belief, because there can for instance not be a goal of flying if the agent does not believe it can fly. Intentions are commitments the agent takes towards achieving a desire (8).

2.2 Changing perspective in software development

Complex and changing environments create new software demands. This report focuses on the oil production domain and has to deal with following challenges (3):

- The oil production domain is complex and dynamic, and monitoring will cause large continuously amounts of different data.
- Increased instrumentation and the use of "smart" well technology will generate a large amount of data to be utilized in the control room.
- To meet local- and global constraints in oil production. Sub-optimization does not necessarily lead to globally optimization.
- To have actions being effective within a certain time limit.
- Difficult and complex interpretation of data to support improved decision-making.
- Detection of potentially dangerous situation.

The challenges mentioned can be derived into motivational factors to create new types of software. Ølmheim, Landre, & Quale (2) pointed out three main three motivational factors: *decreased information load on the human operator, shared situation awareness between technical experts*, and *mutually-agreed timeframe for action*. Lack of shared situation awareness can for instance cause a less optimal[4] oil production. The plans and long term forecast of an optimal solution might not be followed because of different understanding on how to put the plans into operation. Priorities and perceptions about what is critical may differ as well (2).

The existing and traditional generation of software systems, are more or less centralized decision making systems (1). This support typically one kind of equipment, and are mostly data-driven. The use of intelligent software agents addresses these issues. These systems are capable of handling large amounts of data in a physically distributed environment, and have the ability to make autonomously local decisions by reasoning about these data (1). They are capable of handling changes in the environment and react to them continuously. Software agents can also work together with the existing systems (1).

Teamwork is an approach that possibly can create some of the claimed benefits software agents have. The applicability and suitability of teamwork is explained in this chapter, and in the former depth study made by the author of this report (3). This depth study claims that teamwork is well suited for decision support in the oil production domain. Teaming can be divided into agent-centric

[4] Optimization is about maximizing the oil production in the long term, taking the extraction of oil in reservoir into consideration.

and human-centric, described in Section 2.3 and Section 2.4. Research in teamwork focuses on moving towards human-centric system and to include human in the team processes. The paradigm shift is shown in Figure 1.

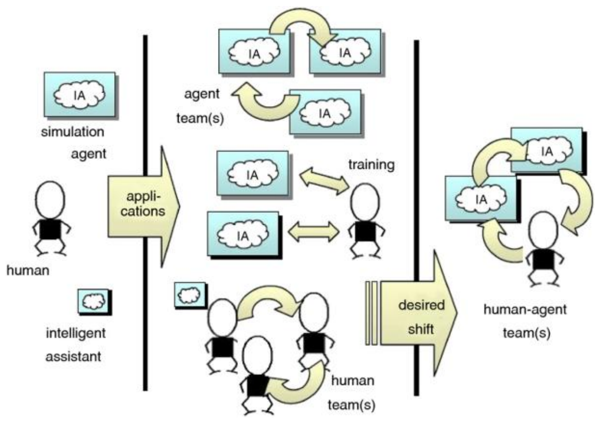


Figure 1: Human - agent teaming (6)

Relevant for both agent-centric and human-centric teaming is the the system's computational elements. This is an important element in complex and dynamic environments with many psycial elements. A type of system called *cyber-physical system* (CPS) features a tight combination and coordination between the system's computational and physical elements (9). This is often referred to as embedded systems, but differs from traditional embedded systems. The elements involved will interact with eachother instead of being standalone devices. For example, many wireless sensor networks monitor some aspect of the environment and forward the processed information to a central node. This type of system focuses on important aspects relevant to the oil production domain. Teamwork can be used to coordinate and create cooperation between the physical components. The distribution of computational elements on different abstraction levels are looked into and realized through the experiment documented in this report.

2.3 Agent-centric system

The teamwork performed in agent-centric system depends on the *delegation of autonomy* between agents. The *degree* of autonomy is determined by how the agent achieves its goals and the authority it is willing to delegate in order to complete the goals. The delegation of autonomy between agents plays an important role in the work documented later in this report, because it says something about the distribution of computation. The delegation of autonomy also influences the achievement of having either a local- or global optimization when solving a problem. These factors are important in the construction of the system design described Chapter 6.

Agent to agent degree of autonomy is shown in Figure 2.

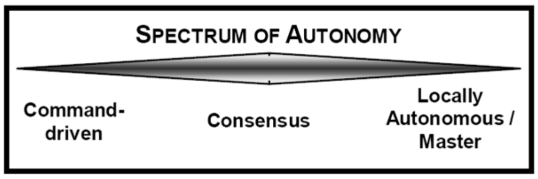


Figure 2: Agent - Agent autonomy taxanomy (10)

An agent’s autonomy increases from left to right shown in the figure. The categories are:

- *Command-driven:* The agent does not plan and must obey orders given by another (master) agent (11).
- *Consensus:* The agent serves as a team member, sharing planning decisions equally with other agents (11).
- *Locally Autonomous / Master:* The agent plans alone and may (if master) or may not give orders to other agents (11).

2.4 Human-centric systems

This section defines what human-centric is and runs through some important aspects related to it.

2.4.1 Overview

Research in teamwork focuses on the human-machine interactions, and how to get humans included in teamwork (5). The effect will be increased situation awareness for humans dealing with unknown or hostile environments (12). It can also help to decrease the information load on the human operator and to arrange mutually-agreed timeframe for action. Ølmheim, Landre, & Quale (2) examined the involvement of humans in the oil production domain which will most likely benefits from such research.

Human-centric system possesses the ability of having the agent system to learn from humans. The learning process enables agents to become so-called *human-centric smart agents* (5). This kind of system can have interaction in both directions. Agents can advise humans, and humans can advise and influence the agent’s reasoning process. Shared plans and joined intentions between humans and agents is something that should be emphasized in a human-centric system.

The following subsections describe how autonomy can be delegated between human operator and the agent system, benfits by using human centric systems, and principles to the creation of such systems. These elements play an important role in the work of creating shared plans and joined intentions between humans and the agents. The mutual learning process that exists in human-centric systems is not considered in this report.

2.4.2 Delegation of autonomy

The delegation of autonomy describes how human operators are involved in the teamprocesses. The *degree* of autonomy is determined by how the human operator chooses to achieve his/her goals, and the authority he/her is willing to delegate in the process of reaching the goals.

The degree of autonomy that a team or teammember have should be able to vary during runtime. Different system states and environmental situations should require different types of autonomy delegation. The trust the human operator has to the system at the moment should reflect the level of delegated autonomy. The degree of autonomy should be based on automatic escalation or de-escalation, and the human operator should be able to change it manually. Learning can build a trustrelationship between the human operator and the agent system. Learning is therefore an important element of escalating or de-escalating the level of autonomy. The learning process should be performed in both directions, causing the agent system to have a chance to learn from the human operator and opposite.

Different degrees of autonomy cause different actions to be made by the human operator and the agent system. Table 1 shows a taxonomy of delegated autonomy defined by PACT-levels (Pilot Authority and Control Tasks). Related to an oil production system, the pilot could be a production engineer, while the computer would represent for instance a field manager or other agents contained in an oil production system.

PACT Locus of authority	PACT Level	Sheridan & Verplank Levels of HMI
Computer monitored by pilot	5b	Computer does everything autonomously
	5a	Computer chooses action, performs it and informs human
Computer backed up by pilot	4b	Computer chooses action and performs it unless human disapproves
	4a	Computer chooses action and performs it if human approves
Pilot backed up by computer	3	Computer suggests options and proposes one of them
Pilot assisted by computer	2	Computer suggests options to human
Pilot assisted by computer only when requested	1	Human asks computer to suggest options
Pilot	0	Whole task done by human except for actual operation (autopilot)

Table 1: Human - Agent autonomy taxanomy (13)

2.4.3 Principles

One should distinguish humans and agents when implementing a human-centric system. Humans and agents are not comparable, but they are complementary in a human-agent team.

Principles made by Tweedale et al. (6) describe how humans and agents are complementary, and are listed as follows:

- Humans are responsible for the output in human-agent teams
- The humans have the mainresponsibility and is therefore in command
- The humans must be activiliy involved in the team processes
- The humans must be adequate informed
- The humans must be able to monitor agent behavior
- The agents' activities must be predictable, so the humans can monitor their activities
- The agent must be able to monitor the performance of the human.
- Each team member (humans and agents) must have knowledge of eachother's commitments towards achieving a goal (intentions).

The principles presented serve as a foundation in the construction of the objects to be study in this report. The system design that reflects some of the principles can be found in Chapter 6.

2.4.4 Benefits

The BDI architecture constructs a human-like intelligence (6). This enables the agents to substitute humans. The disadvantage is that the human-like substitute could fail at a critical point without leaving any choice to the human for regaining control of the situation. The results will be impaired situation awareness. Inclusion of the human operator in the team-processes leads to human-centric systems and shared situation awareness (5).

Interaction between machine and human in human-centric systems enables customized decision support in the decision making process. Interaction does not only consist of a communication language, but adds the elements of *observation* and *adaption*. Truly smart agents can be *complementary to a human* by adopting skills similar to a human, and may include communication, learning and coordination, rather than being a simple replacement to a human (5). Learning is not discussed in this report. Communication and coordination are described in Subsection 2.5.1.

Human-centric systems make use of delegating autonomy between human and agents. The delegation of autonomy describes how human operators are involved in the teamprocesses. Lucas & Shepherdson (14) list the following advantages moving towards systems with an architecture based on delegation of autonomy:

- *Lines of authority and communications* are clearly defined. This makes surrounding comprehensible.
- *Decision-making is delegated* where possible, and ensures that workload is shared appropriately among the managers and members of the organization. Delays resulting from a too high a workload are then minimized.
- Greater *responsiveness*, because decision do not always need to go to the top of an authority hierarchy.
- Improved *communications*. The involved parties only receive information necessary for the role, causing less irrelevant and distracting details.
- *Decisions are made at the appropriate level* in the organization, because of already established paths of authority and criteria for escalation.
- *Productivity* is higher as the agents are able to make many decisions locally, causing less waiting time for decisions.
- Reduced workload on *Human operators* that could help the operator to focus on the critical situations, which could lead to better decisions and less mistakes.
- Creates a *higher level of local decisions,* and avoid "bad" decisions propagate through the system.
- Enable *distributed reasoning* which creates feasibility of parallel computations and modularity causing better and easier maintenance of the system. The parallelism could help creating a more scalable system.

2.5 Teamwork

This section clearifies the difference between teamwork and collaboration between agents. The section also describes coordination techniques, team variables, and challenges in teamwork.

2.5.1 Collaboration and teamwork

Agents' social ability makes them able to communicate with eachother. This ability can be used in their achievement of goals. When achieving goals in groups, they will achieve more than they can do as individuals. This is analog to human groups. They will get a lot more done if they work together. They can share information and work in a coordinated manner, and make eachother more efficient and competent (12). There are two possible ways agents work together: collaboration and teamwork.

Collaboration is to *communicate*, *cooperate*, and *coordinate*. *Coordination* is referring to agents that are freely allowed to communicate and enforce agreements prior taking decisions (15). These agreements do not have to work towards a common goal, but can be an agreement as a result of negotiation. The agents are voluntariliy entering the relationship with eachother to achieve a system derived goal (16). *Coordination* is the ability to manage the interdependencies of activities between agents (16). Coordination prevents for instance two soccerplayers in kicking the ball at the same time. An agent can also coordinate its actions with another agent unaware of its presence. Coordination does not imply cooperation (17). Communication is interteraction, typically a two-way process, where all agents can potentially be senders and receivers of messages (15). Communication can be used for coordination among cooperative agents or for negotiation among self-interested agents (15).

Teamwork is a more structured type of collaboration. It is more than agents communicating and acting in a simultaneous and coordinated manner, or agents asking for and providing services to eachother (12). Teamwork has one additional requirement. The team of agents works towards something together, such as an *achievement of a team goal* (12). A team can be defined as a set of agents that has a shared objective and a shared mental state (18). The aims of agent teamwork research are to improve the concept understanding, to develop some reusable algorithms, and to build high-performance teams in dynamic and possibly hostile environments (18). Coordination and cooperation are necessary for agents in a team to achieve a common goal (18).

An example can help to illustrate some of the differences between collaboration and teamwork (12). The car traffic has cars that work as autonomously units trying to avoid crashes and to reach their destinations. In order to do this, they have to avoid eachother, stop for eachother, etc. They will have to coordinate their actions to achieve their goals and finally reach their destination. If a car breaks down, they will just drive by it and keep on going to they reach their destinations. For a convoy represented as a team, things are a bit different. Their goal is to reach the destionation together. If a car breaks down in the convoy, all cars in the convoy are affected by it. They want to achieve their team goal, but cannot do so if one car breaks down. If a car breaks down they have to fix the car or change it, in order to achieve their team goal.

2.5.2 Coordination techniques

Coordination is required when agents are interdependent, for example, when agents share tasks or avoid resource conflicts (19). Many approaches exist in the work of coordination of agent systems. Nwana, Ndumu, Lee & Collis (20) define four broad categories that will be presented in this subsection. The four categories are called organizational structuring, contracting, multi-agent planning, and negotiation.

The first technique is named *organizational structuring.* This is a coordination technique that exploits the structure of the society, the role of the different agents; and their relationships with each other. This is for instance a client-server system. Another type is the master/slave coordination approach. Here, the master generates the plans, and distribute fragments of the plan to the slaves. Conducting the plans, the slaves have to report to the master. The master then has full autonomy and the slaves have only partial autonomy (21).

The second technique is named *Contracting*. First the manager agent announces a contract, receives bids from other agents, evaluates the bids, and finally awards the contract to the winner. The contract-net protocol is a typical contracting technique. Other various auction protocols exist (21).

The third technique is named *Multi-agent planning.* This coordination technique resolves any foreseen conflicts between the agents' plans. There are two types of multi-agent planning, which are described as:

- *Centralized multi-agent planning:* A central agent performs planning on behalf of the society. It receives receipts of all partial or local plans from individual agents, and analyzes them in order to find potential inconsistencies and conflicting interactions. Next, the planning agent modifies the partial plan and combines them into a multi-agent plan, without inconsistencies and conflicting interactions (21).
- *Distributed multi-agent planning:* the agents exchange partial sub plans which progressively build the multi-agent plan without inconsistencies and conflicting interactions (21).

The fourth technique is named *Negotiation.* Nwana, Lee & Jennings (21) defines negotiation as following: *"...negotiation is the communication process of a group of agents in order to reach a mutually accepted agreement on some matter."* This agreement take place after a dialogue between the parties, where they exchange proposals with each other, evaluate the proposals, and exchange counterproposals until an agreement between the agents is reached.

2.5.3 Team variables

Different aspects of teamwork can vary from system to system. The team variables identified here will have different focus later in the report. They have different importance and variations, as we will see later in the implementations of the reference problem defined in Chapter 5. The team variables identified are:

- *Team size:* Number of agents involved as teammembers.

- *Team structure:* Formation of the team. "Flat" structure with all agents on the same level, or hierarchical structure where agents are situated on more than on level. In order to achieve different goals at specific conditions, different team formations are needed. The team life cycle (12) to achieve a goal is as follows:

    1. Team formation:

        a) Find potential team members
        b) Recruit team members

    2. Team task execution:

        a) Decompose task into sub-tasks

    b) Distribute/delegate sub-tasks to appropriate team members
    c) Each team member performs its allocated task in a coordinated fashion

3. Disband team:

    a) Each member leaves and is no longer committed to the team

- *Differentiation and specialization:* A hetergenous team of agents contains agents that can fulfil only specific roles. To repair such a team by using the team's member is not possible. To solve this problem, an agent from outside the team has to fulfil the role, or else the team will fail. A homogenous team contains agents that can fulfil different kind of roles, and can therefore possibly repair it if an agent has failed.

- *Failure handling:* Who shall handle the failure if a team fails? The failure can be handled by trying another approach to achieve the goal, or the incapsulated team has to handle the failure.

- *Authority delegation:* This aspect can be divided into the two cateogories of machine to machine delegation, and human to machine delegation. The delegation looks at how the machine or human is involved in teamprocesses according to authority they are delegated.

2.5.4 Challenges in teamwork

Tweedale et al. list three primary challenges (5) that teamwork should overcome: *communication*, *negotiation*, and *trust*.

Communication is the first challenge. It enables agents to understand eachother. The communication must be efficient and robust to recover from errors, and provide the possibility of asking and providing services. Facilitator agents can provide matching the available services and request. Another approach is mobile agents that can move from one environment to another. A third approach is hierarchical structures that will lay constraints on how to communicate with eachother.

Negotiation is the second challenge. Teams have requirements, and so has the individual agents. A negotiation process is needed for the team to achieve it goals, and at the same time have the individual agents having their autonomy.

Trust is the third challenge. This is for example if an agent should trust another agent receiving correct information, or that the agent can perform a particular task. This is not easily measured, but should is reflected on the delegation of autonomy.

Chapter 3
Framework and tools

This chapter introduces the framework and development tools used in the work of conducting the experiment constructed in Chapter 4. The development tools are used for system design and implementation of the system used in the experiment. The framework tested in the experiment contains the modelling paradigms JACK Agents and JACK Teams. The framework is described in Section 3.1.

3.1 Modelling paradigms compared

The two modelling paradigms JACK Agents and JACK Teams were used in the work documented in this report. They were used to develop a system, which resulted in two versions of the same system so they could easily be compared. The comparison shows the applicability and suitability for the two modelling paradigms regarding teamwork, which is the main objective of this report.

3.1.1 Overview

JACK Agents has been developed to provide agent-oriented programming. JACK Teams is an extension to JACK Intelligent Agents that provides a teamoriented modelling framework (22). JACK Agents and JACK Teams are extensions to the Java programming language. JACK Agents and JACK Teams source code is first compiled into regular Java code before being executed.

3.1.2 JACK Agents

The JACK Agent Language introduces six class-level constructs. These constructs are:

- *Agent:* The agent-construct is used to define the behaviour of a BDI[5] intelligent software agent. This includes the agent's capabilities, what type of messages and events it responds to/sends, and which plans it will use to achieve its goals (23). The agent provides reasoning behaviour under both proactive and reactive stimuli (24).

- *Capability* – The capability-construct is used by an agent to aggregate and reuse functional components to give it certain abilities (23). A capability can be made up of plans, events, beliefsets, and other capabilities (23).

- *BeliefSet:* The beliefset-construct represents agent beliefs using a generic relational model (23). The beliefset is designed in a way that allows it to be quieried about, using logical members. Logical members are like normal data members, except that they follow the rules of logic programming (like Prolog for instance) (23).

- *View:* The view-construct allows general purpose queries to be made about an underlying data model (23). The data model can integrate a wide range of data sources such as JACK beliefset, Java data structures, and legacy systems (23).

[5] Belief-Desire-Intention

- *Event:* The event-construct is an occurrence that triggers the agent to take some sort of action. There are several types of events. All types of events can be carriers of goals that the agent shall try to achieve (2). The key difference between the two main categories normal events and BDI events is how an agent selects plans to execute. At a conceptual level, the BDI reasoning models goal-directed behaviour in agents, rather than plan-directed behaviour (23).

- *Plan:* The plan-construct generates instructions the agent follows to respond to an event received. The plans are analogous to functions. First, a check is done to determine if the plan is relevant for exactly that specific instance of the event. Checking for relevance provides the agent with a filter to exclude plans that will definitely not be able to handle the event. If the plan is relevant, an applicability check takes place. To check for applicability, the current circumstances (values of respective members and data structures) and the agent's current beliefs (represented by its beliefset relations) are evaluated to check if the plan is applicable to the current conditions. The plan will execute its steps of action if it is relevant and applicable (23).

Reasoning statements are JACK Agent Language specific statements that can only appear in reasoning methods. They describe actions that the agent can perform to execute behaviour. Actions such as posting events, sending messages to other agents or waiting until a particular condition is true are expressed using reasoning method statements. The important communication reasoning method statements (@-statements) between agents, and emphasized statements due to teamwork are:

- *@send(agent_name, message_event):* The @send statement is used to send a message event to another agent from within a reasoning method (23).

- *@reply(original_event, reply_event):* The @reply statement is used by an agent to reply to a message event that it has received from another agent (23).

3.1.3 JACK Teams

The JACK Teams extension introduces the new constructs team, role, teamdata, and teamplan. The JACK Teams model includes all the programming contstructs contained in the JACK BDI Agent model, but with an extended semantics for some constructs. Note that an agent-instance has to be renamed to be a team-instance when it is compiled using JACK Teams. The behaviour of the newly created team-instance will still act as an agent-instance, because a team-construct is only an extension of the agent-construct. The extendend and additional constructs are:

- *Team:* The team-construct is an extension of the JACK Agents's agent-construct. This reasoning entity is characterised by the roles it performs and/or the roles it requires other teams to perform (22). Attaching subteams capable to fulfil the required roles create the team formation. Teams and subteams are both made from the team-construct. Subteams may also require roles to be fulfilled, and will cause a hierarchy (ies) of roles as a result. Note that a subteam can fulfil more that one role at the same time. The team are automatically provided with objects to hold the actual role/sub-team selections. These objects are known as role containers (22).

- *Role:* The role-construct defines a relationship between teams and subteams. The role relationship is expressed in terms of the event and belief exchanges implied by the relationship (22).

- *Teamdata:* The teamdata-contstruct is similar to the JACK Agents's beliefset, but contains the ability for belief propagations in addition. The propagation is going in both directions between team and subteam, so-called belief propagation and belief inheritation. The use of teambelief, in addition to the team coordination statement (see reasoning statement @teamAchieve), enables sophisticated team behaviours to be implemented (22).

- *Teamplan:* The teamplan-construct is an extension of the JACK Agents's plan-construct. A teamplan specifies how a task is achieved in terms of one or more roles (22). The process of determine what teams to perform the different roles are known as team formation in the framework. The behaviour specified in terms of roles decouples the team's behaviour from the subteam's behaviour (22). It is however possible to perform reasoning based on the actual team membership if needed, because the team can access its possible sub-teams through the role container.

- *Initialisation file (**not** a JACK entity-construct):* The file is used to build the so-called role obligation structure (22). The overall lifetime of a team has two phases (22). The first phase is for setting up an initial role obligation structure. That is the declaration of which teams are capable to perform which roles for the specific teams. The second phase constitutes the actual operation of the team during runtime to solve a specific task. The first phase is handled by the initialisation file, which is generated in JACOB[6] format. Note that it is possible to modify this structure during runtime (22).

Reasoning statements are JACK Teams Language specific statements that can only appear in reasoning methods. They describe actions that the team can perform to execute behaviour. Additional reasoning method statement (@-statements) compared to JACK Agents, which is an important statements due to teamwork:

- *@teamAchieve Statement:* The @teamAchieve statement is used to activate a sub-team by sending an event to the role fulfilled by the subteam. The team that sent an event through the @teamAchieve statement waits until the event has been processed by the sub-team. If the event succeed or fails, so will the @teamAchieve statement. In combination with the JACK @parallel statement, a wide range of team behaviours can be implemented (22). The @parallel statement is the same as contained in JACK Agents, and is therefore not used during the work documented in this report.

3.2 Previous experiences

This section describes previous experiences made by using JACK Agents and JACK Teams.

[6] The JACOB™ Object Modeller (JACOB) is a system providing machine and language independent object structures (42).

3.2.1 JACK Agents

StatoilHydro has a relatively long history with multi-agent systems and JACK Agents (2), described in the paper made by Ølmheim, Landre, & Quale (2). The paper shows for one thing the suitability of JACK Agents in the development of an oil production system.

The domain is the same as the reference problem defined in this report, and will therefore be a contribution in the shaping of the system design described in Chapter 6.

3.2.2 JACK Teams

Jarvis et al. emphasize the following advantages by using JACK Teams in their work (25):

- Behaviour in the different teams is clearly separated. This makes it easier to change components, since the behavior is defined according to roles, and not teams. How different teams fulfil the same role, is therefore not important. Teams can therefore be replaced, as long as they fulfil the role they were set to do.
- The plan describing the steps actions is written in terms of roles, not specific sub-teams, thus making the plan resource independent.
- Behaviour of teams can be understood independently because plans are written in terms of roles.
- The role concept in JACK Teams enables team structures and behaviours to be specified independently of the eventual members of the team. Thus it provides the flexibility for team formation to occur dynamically and in response to changing circumstances.
- A team is able to subtask its sub-teams and propagate beliefs to its sub-teams through the role instances. If required, the actual sub-team instances that are available to perform a role are accessible through the role obligation structure.

Bisht et al. (26) are using JACK Teams in the simulation of battlefields, used by the military. This paper mentiones the following experiences made:

- JACK Teams gives a clear and concise description of coordinated activities and allows the abstraction of what needs to be done from how it is done, i.e., the responsibilities of the team can be written down without consideration of how the roles would be fulfilled and implemented by the team members.
- A relatively simple team programs become complex. For example, to implement this scenario, the authors had 4 agent files, 3 capability files, 6 team files, 4 role files, 37 plan files and 25 event files.
- The resultant code was highly modular and maintainable, which would not have been possible otherwise.

Cheong (12) describes two different kinds of Collaborative Agent Architectures which provides support for agent teamwork: Yellow Pages service to locate agents by their capabilities, a Facilitator agent to facilitate agent coordination or a team class which can be extended to create agent-teams. A team class will function like JACK Teams with its team-construct.

A Facilitator agent coordinates tasks for the multi-agent system. When all agents start up, they are required to register with the Facilitator agent. Registration involves informing the Facilitator agent of the tasks that they can perform. When an agent requires a service from the community (i.e. the team), it sends a query to the Facilitator. The Facilitator agent uses agent registration details to determine which agents can help to solve the query. The Facilitator delegates the tasks to all agents that can help solve the query. They will then perform the tasks and return the results to the Facilitator agent. The Facilitator then sorts all the results and returns them to the Requesting Agent.

This report documents two systems developed to perform as objects of study in the experiment designed in Chapter 4. The first system constructed uses JACK Agents to construct the structure with a Facilitator agent described above. The second system constructed uses JACK Teams and with a team class as described above.

The paper written by Daren (27) contains a comparison between JACK Agents and JACK Teams. This project used a JACK Agents implementation of two agent-instances that communicated and coordinated with eachother. The JACK Team solution had a team-instance coordinating all communication between the two agents-instances. The question is if an extra agent-instance could have performed the same type of coordination if it substituted the team-instance. To build the system with an agent-instance substituting the team-instance could resulted in the same flexibility and correctness. This remark is also noted in the paper.

3.3 JACK Development Kit

The JACK Development Environment (JDE) is a cross-platform graphical editor suite written entirely in Java for developing JACK agent and team based applications (28). The JDE is a toolkit that allows construction of detailed design, construction of JACK entities, and it supports reuse of components (28). The editor uses drag-and-drop to to create detailed design. The editor generates a skeleton of JACK code derived from this design. The JACK compiler compiles the JACK code into regular Java source code before execution (28).

JDE was used in the work of designing the JACK Agents and JACK Teams systems to be made, due to the possibility of graphical modelling for both of them. The sourcecode was written in both JDE and Eclipse IDE (see Section 3.4). The JACK Agents solution was implemented in Eclipse IDE, and the JACK Teams solution in JDE. The original plan was to use Eclipse for both modelling paradigms, but the plugin for the JACK framework in Eclipse did not work properly for JACK Teams.

3.4 Eclipse Integrated Development Environment

Eclipse Integrated Development Environment (IDE) is a software platform comprising extensible application frameworks, tools, and a runtime library for software development and management (29). What most people associate with Eclipse IDE is Eclipse's Java development environment (30). The Eclipse extensible software framework allows users to extend its capabilities by installing and writing their own plugins (29).

Eclipse IDE was selected to be the code-editor instead of JDE (see Section 3.3), because it is familiar for the author, and it supports incremental code compilation. A plugin was installed in Eclipse IDE to support development using the JACK framework. This worked only properly when JACK Agents was used, and not with JACK Teams. JDE was therefore used to develop the system in JACK Teams.

Part II

Own contribution

Chapter 4

Approach

This chapter describes the approach this project uses to compare the JACK Agents and the JACK Teams modeling paradigms. The approach is to conduct an experiment designed in this chapter. The approach is divided in two parts. The first part is a quantitative approach. The second part is a qualitative approach that will give more depth in the comparison, in addition to the quantitative approach. The result of the approaches is described in Chapter 9 and Chapter 10.

4.1 Quantitative approach

Experiments are used when we want control over the situation and manipulate behaviour directly, precisely, and systematically (24). An experiment is performed in order to be able to decide empirical that one method is better that the other. This inspection involves use of methods for statistical inference with the purpose of showing with statistical significance that one method is better than the other (31). To carry out an experiment, several steps of construction have to be followed. The process to conduct an experiment contains the steps of defining, planning, operation, analysis and interpretation, presentation and package. The following subsections will elaborate these steps.

4.1.1 Experiment type

The results of the experiment are evaluated quantitative by using statistical analysis to to draw conclusions. The experiment conducted in this report is of type quasi experiment. It cannot be called a true experiment, because it is impossible to perform random assignment of the subjects to the different treatments. A treatment is one particular value of a factor, modelling paradigms (JACK Agents and JACK Teams) in this case. The subjects to be evaluated in this experiment are two implemented solutions of the same type of oil production system.

4.1.2 Experiment process

The steps in the experiement process are suggested by Claes Wohlin et al. (31), and are described as follows:

- *Experiment definition:* The foundation of the experiment is determined by its definition. The purpose is to define the purpose of the experiment in terms of the objective, purpose, quality focus, perspective, and context.
- *Experiment planning:* The experiment definition explains why the experiment is conducted. The planning prepares how the experiment is conducted. This step determines the context, state hypotheses, design experiment, and evaluates possible threats.
- *Experiment operation:* The subjects are prepared and made ready for evaluation. The experiment is executed and data is collected.
- *Analysis and interpretation:* Measured data is gathered in statistical analysis.

The steps above will serve as guidelines in how this experiment will be conducted, to ensure good quality on the findings. The definition step and operation step will be handled in the following subsections. The operation, which is the system development, will take place in Chapter 7 and Chapter 8. The analysis and interpretation will find place in Chapter 9.

4.1.3 Experiment definition

Before planning and execution takes place, the experiment needs to be defined. The purpose of the experiment definition is to ensure that importart aspects of the experiment are defined. This is done through using the *"Goal Question Metric" (GQM)* template described by Claes Wohlin et al. (31). The goal template is:

> 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 different elements in the template are related to the experiment contained in this report as follows:

- *Objects of Study:* The objects of study are the entities that are studied in the experiment. The experiment conducted in this report will have two objects that are the two different solutions made by using JACK Agents and JACK Teams.
- *Purpose:* The purpose defines the intention of the experiment. The intention of the experiment will in this report be to look the applicability and suitability of JACK Agents and JACK Teams when constructing teamwork. The author believes that the applicability and suitability can be measured and evaluated by having a finished JACK Agents version converted into a JACK Teams version, only changing the necessary code to convert the application using JACK Teams constructs.
- *Quality Focus:* This is the primary effect being studied in the experiment. Quality focus in this experiment will be applicability and suitability of teamwork.
- *Perspective:* Perspective is the viewpoint the results are interpreted according to. This experiment has the perspective of a software developer.
- *Context:* The context is the environment in which the experiment runs. The environment in this experiment consists of the personell (subjects) involved in the experiment and the software artifacts (objects) involved. The author (a student with experience at university level) will be the subject. The objects of context are the JACK framework containing the two modeling paradigms JACK Agents and JACK Teams and the application domain that is decision-support systems used in oil production.

The definition for the experiment then turns out as follows:

> Analyse two different implementations of an oil production system
> for the purpose of evaluation
> with respect to applicability and suitability of teamwork

from the point of view of software developers
in the context of the student using JACK Agents and JACK Teams to implement the system.

4.1.4 Experiment planning

This subsection will handle the planning phase. The planning prepares for how the experiment is conducted. The context selection selects the environment in which the experiment will run. The next two steps are to formulate hypotheses and select variables to look at in the experiment. These step are followed by the selection of subjects, experiment design, and validity evaluation.

Context selection

A specific reference problem of application domain that is decision-support systems used in oil production shall be implemented in two versions. One made by using JACK Agents, and the other using JACK Teams. A comparsion will be based on the two versions, doing the same work, but in a different way. The two versions are as follows:

- *JACK Agents version:* Physical components from the reference problem in Chapter 5 are represented as agents. The design of the system can be found in Chapter 7. The JACK Agents paradigm is described in Subsection 3.1.2
- *JACK Teams version:* Some of the physical components from the reference problem in Chapter 5 are represented as team-instances. The rest of the system will remain as agent, same as in the JACK Agents version. The design of the system can be found in Chapter 8. The JACK Teams paradigm is described in Subsection 3.1.3.

Both versions have the same graphical user interface (GUI). The two different versions will be compared to discover differences to find their possible applicability and suitability in teamwork, used in a system designed (see Chapter 6) based on the reference problem (see Chapter 5).

The context can be characterized according to four dimensions (31). The four dimensions are listed with explanations as follows:

- *Offline vs. Online:* The experiment conducted is offline. The devolped systems will not be deployed in a real oil field, only as simplificated simulated oil production system.
- *Student vs. Proffesional:* The two solutions developed are constructed by the student that made this report.
- *Toy vs. Real problems:* The oil production system designed does not reproduce the complexity a real oil production system has to handle. The main cause-effect relationships of the challenges addressed is however maintained in the solutions.
- *Specific vs. General:* The experiment is made specific for oil production systems which are built in a similar hierarchical structure as designed in Chapter 6, based on the reference problem. The findings can possibly indicate applicability and suitability in similar domains with the same hierarchical structure.

Hypothesis formulation

Hypothesis testing is the basis for the statistical analysis of the experiment. A hypothesis is stated formally and comfirmed or rejected by data collected.

To evaluate JACK Agents and JACK Teams applicability and suitability regarding teamwork in an oil production system. To address this evaluation and comparison between the two versions, Table 2 contains several hypotheses to evaluate this. The null hypothesis states that there are no real underlying trends or patterns in the experiment settings (31). The alternative hypothesis in the table is the hypothesis in favour of which the null hypothesis is rejected (31). The formulation of the hypothesis is done on background of the problem definition formulated in Chapter 1. Some of the hypotheses formulated require objective measure, a value dependent only from the measured object. Hypothesis 6 requires subjective measure, which depends on human judgement.

Id	***Hypothesis***
H01	The functionality of the the two versions will be implemented with approximately the same number of code lines
HA1.1	The JACK Teams version will implement the same functionality as JACK Agents version with fewer lines of code.
HA1.2	The JACK Teams version will implement the same functionality as JACK Agents version with more lines of code.
H02	The number of entities will be the same for the two oil production system versions.
HA2.1	The JACK Teams version will have more entities than JACK Agents version.
HA2.2	The JACK Teams version will have fewer entities than JACK Agents version.
H03	Both versions will use the same number of functions to complete the designed oil production system given in Chapter 6.
HA3.1	The JACK Teams version will complete the designed oil production system (see Chapter 6) with fewer functions than JACK Agents version.
HA3.2	The JACK Teams version will complete the designed oil production system (see Chapter 6) with a larger number of functions than JACK Agents version.
H04	Both versions will have the same number of couplings between the components in the system.
HA4.1	The JACK Teams version will have fewer couplings between the components than JACK Agents version.
HA4.2	The JACK Teams version will have more couplings between the components than JACK Agents version.
H05	JACK Agents version and The JACK Teams version have the same number of external operations changing their internal state.
HA5.1	The JACK Teams version has a fewer external operations changing the internal state than JACK Agents version.
HA5.2	The JACK Teams version has larger amount of external operations changing the interna state than JACK Agents version.
H06	Use of JACK Teams will not provide a higher abstraction level for modeling and implementation of teamwork in an oil production system, compared to JACK Agents.
HA6.1	Use of JACK Teams will provide a higher abstraction level for modeling and implementation of teamwork in an oil production system, compared to JACK Agents.

Table 2: Formulated hypotheses

Variable selection and experiment design

Variables are divided into independent- and dependent variables. Independent variables are variables that we can control and change in the experiment. The dependent variables are variables that can be measured to see the effect of the treatments (one particular value of a factor).

This experiment consists of one factor, which is the oil production system. Further, two treatments are compared to eachother. The treatments are two versions of the oil production system, the JACK

Agents version and the JACK Teams version. The dependent variable is the software engineering regarding teamwork between the physical components represented in the oil production system. The dependent variable from the two different versions will be compared in order to choose the appropriate hypotheses from Table 2.

Validity evaluation

The validity threats should be examined in the early planning phase in order to address them in a satisfying manner. Adequate validity is that the results should be valid for the population of interest. First, the results should be valid for the population from which the sample is drawn. That is the specific oil production system design in this report (see Chapter 6). Secondly, the results may be in interest to generalize about in order to be valid for the whole population. The population in this experiment is defined to be oil production system in general.

This experiment will use a scheme from Cook & Campell (28) with classification of four threats. The classification scheme consists of the categories conclusion-, internal-, construct- and external vailidity (31). Figure 3 shows how the different categories relate to the different parts of the experiment process.

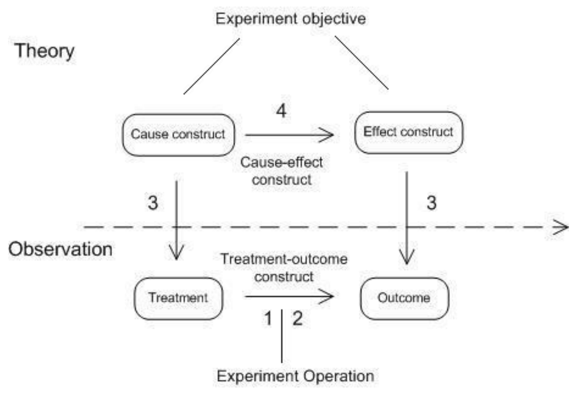


Figure 3: Experiment principles (31)

The figure is divided into a *Theory area* and an *Observation area*. The observations made in the experiment shall conclude the theory in the hypothesis. The different categories of validity threats are described as follows (31):

1. *Conclusion validity:* Concerned with the relationship between the treatment and the outcome. There shall be a statistical relationship with a given significance.
2. *Internal validity:* Concerned with observed relationship between the treatment and the outcome. The treatment shall cause the outcome, and not by some factor that is under no control or by a factor that has not been measured.
3. *Construct validity:* Concerned with the relationship between the theory and observation. If the relationship is causal, the treatment reflects the construct of cause and the outcome reflects the effect construct.
4. *External validity:* Concerned with generalization. If there is a causal relationship between the construct of the cause and the effect, external validity is if the results can be generalized outside the study achieving the same results.

Threats identified in this experiment are divided into the different categories mentioned above. The threats to category 1, *conclusion validity* are:

- *Low statistical power:* The power of the statistical test is the ability of the test to reveal a true pattern in the data. The two systems implemented in this experiment does not contain the size of a physical components involved in realistic production system. When constructing the JACK Teams version, not all agent-teams are replaced with teams-instances from JACK Teams, but they are used in a combination in the JACK Teams version. We accept this risk and take it into consideration that some of the conclusions made can be erroneous.

- *Fishing:* The experiment performed is a quasi-experiment. This means that the evaluated objects are not selected randomly. When constructing the two versions, no specific outcome should be in mind when constructing and implementing the two versions to get a desired result.

- *Reliability of measures:* When measuring a phenomen twice, the outcome shall be the same. Human judgement can affect the reliability, since they do things different from time to time. This threat is addressed by using metrics that involve a small degree of human judgement.

The threats to category 2, *internal validity* are:

- *Maturation:* The effect of that the subjects react differently as time passes. The two versions of the program will be developed at different times in the work of this project. The threat is addressed by developing the JACK Agents version first, and then converting it into a JACK Teams version. The JACK Team version then reuses the structure and algorithms that functions in both modelling paradigms.

- *Selection:* The effect of natural variation in human performance. There is only one person developing both versions. Coding style and motivation at the development time, and timepressure will play a role according to this threat. This risk is accepted, and addressed by reusing as much as possible between the two versions to ensure a coding style as similar as possible.

The threats to category 3, *construct validity* are:

- *Inadequate preoperational explication of constructs:* This means that the constructs are not sufficiently defined, before they are translated into measures and treatments. The problem definition formulated in Chapter 1 has to be clear on what to expect of results, and the theory on teamwork and the system design should address teamwork with its properties laid out in Section 2.5 as a foundation.

- *Mono-operation bias:* If the experiment under-represent the construct and may not give the full picture of the theory. Only one program of each version is made. The programs are also constructed from a specific design that does not represent the complexity a real oil production system has to handle. We accept this threat and and take into consideration when generalizing the result and findings.

- *Confounding constructs and levels of constructs:* The effect of the presence of the construct is confounded with the effect of the level of the construct. No experience has been made using JACK Agents or JACK Teams before the work of this project was carried out. The use of some abilities the frameworks offer is therefore maybe not used to its full potential, and a higher level of experience would maybe change the coding to be more suited to fit the two different versions. This threat is accepted and is addressed in some degree by using the one version as basis for the development of the other versions. The things that are changed, is the things that can be done in a way that is characterized and mainconstructs in the other framework.

The threats to category 4, *external validity* are:

- *Interaction of selection and treatment:* Generalizing the results, the author may not be representative for the representative population, namely the software developers. The threat is accepted, and the lack of experiences the author has as a student is recognized.

- *Interaction of setting and treatment:* The effect of not having the experimental setting of industrial practice. This threat is address by using development tools and methods that are up to date.

4.1.5 Experiment construction

The experiment construction connects hypotheses and metrics with the benefits that will show applicability and suitability of JACK Agents and JACK Teams doing teamwork in an oil production system.

Benefits

The benefits recognized and examined are the same benefits mentioned for agent technology (32), with emphasizing reduced development effort and high abstraction level when implementing teamwork in an oil production system. Reduced development effort will reduce amount of code, number of entities and functions. The benefits are pointed out as follows:

- *Reduced development effort*

- *Reduced coupling*

- *Encapsulation of functionality*

- *High abstraction level*

Metrics

The experiment is a quantitative research method. The relationship and comparison between the two versions of the oil production system shall be measured and analysed according to the metrics given in this subsection. The following metrics will be used in the evaluation:

- *M1 Lines of Code (LOC):* The number of written code-lines. Code-lines will be counted as the number of semicolons in the source-code.

- *M2 Number of Entities (NOE):* In the JACK Agent solution this will be the events, plans, capabilities, agents, views, beliefset, and Java classes. The addition constructs in JACK Teams are teamplans, roles, and teams.

- *M3 Number of Functions (NOF):* The number of JACK-methods, Java-metods, and plans/teamplans used by the agents and teams and their plans/teamplans.

- *M4 Number of Couplings between Entities (NOCBE):* Couplings in the JACK Agents version will in addition to in- and out going method-calls, be events sent and received by the agents. Couplings in the JACK Teams version will in addition to in- and out going method-calls, be events sent and received by the teams. Method calls from plans to its belonging agent/team and posted events within the agent/team are not counted. Instanciating Java objects and and JACK objects are not counted either.

- *M5 Number of External Activations (NOEA):* External activations in the JACK Agents version will be external method-calls and received events. External activations in the JACK Teams version will be external method-calls and received events.

The relationship between benefits, hypotheses and metrics
There exists a relationship between the four benefits mentioned above, the hypotheses (see Subsection 4.1.4) and metrics (see above). Metrics are used to find the correct hyphothesis to confirm, which will represent different benefits according to which hypothesis that is confirmed. The relationships between the three elements are shown in Table 3. This table will be used to find the wanted results and conclude with findings according to it.

Benefit	**Hypothesis**	**Metric**
Development Effort	H01/HA1.1/HA1.2	M1(LOC)
	H02/HA2.1/HA2.2	M2(NOE)
	H03/HA3.1/HA3.2	M3(NOF)
Reduced coupling	H04/HA4.1/HA4.2	M4(NOCBE)
Encapsulation of functionality	H05/HA5.1/HA5.2	M5(NOEA)
High abstraction level	H06/HA6.1	Qualitative Result

Table 3: Benefits, hypotheses, and metrics

4.2 Qualitative approach

The qualitative approach compares the JACK Agents and JACK Teams solutions developed in the work of the experiment described in Section 4.1. Advantages and disadvantages between the two modelling paradigms are identified and explained. This will give more depth in the comparison of the two modelling paradigms, in addition to the quantitative approach.

Interpretation of the qualitative data is more closely tied to the researcher (their identity, backgournds, assumptions and beliefs) than in quanitative data analysis. This means that their conclusion must be much more tentative than those from quantitative data analysis (33).

The qualitative analysis compares advantages and disadvantages JACK Agents and JACK Teams have compared to eachother, looking at the following aspects:

- *Autonomy:* Lucas & Shepherdson (14) define autonomy as *"the need for decisions to be made at any time, with some appreciation for the circumstance of the current situation (often referred to as situation awareness)."* Delegation of autonomy can be divided into the two cateogories machine to machine delegation, and human to machine delegation. The delegation looks at how the machine and/or human is involved in teamprocesses. The aspect covers challenges related to distribution of reasoning needed and storage.

- *Scalability:* Scalability indicates the system's ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged (34). This quality attribute looks at development effort when expanding the system with more instances of the different agent- or teams constructs.

Chapter 5

Defining a reference problem

This chapter describes a case that is considered to benefit from using teamwork. The problem specified in accordance to the application domain *decision-support systems used in oil production.* The reference problem described will create the foundation for the system design in Chapter 6. The reference problem put restrictions on how teamwork can be included and evaluated in the experiment.

5.1 Oil production system

Hydrocarbon production and processing are processes in production field have to be dealt with on daily basis. The production is concerned with extracting hydrocarbons from the production wells, and processing is associated with separating hydrocarbons from water and sand. The processing also includes the uniting of the production from different wells, in order to maximize market value (3). This report will refer to the production and processing system as oil production system, since oil will be assumed to be the wanted substance. Gas is defined as waste in order to reduce the number of parameters taken into consideration in the system design of the oil production system constructed in this project.

The oil production system is about optimizing the oil production. The human operator plans and monitors the asset with respect to meet the production goals for the asset, and implement the production plan and monitor the process state and the general performance on a continuous basis. Production targets are the amount of oil and the amount of waste. Different oil quality is not taken into consideration in the work documented in this report. Waste is defined to be water, sand, and gas.

The system designed is 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 oil field. Agent-instances and team-instances should be capable of adjusting the production in a globally optimized manner. Global optimizion is performed when the oil field at all times produces with the wells that have the best oil/waste-ratio. This type of optimization is an optimal production, which is defined in Section 5.4 to be maximizing of the oil production in the long term, taking the extraction of oil in reservoir into consideration.

5.2 Reference model

To represent an oil field, Figure 4 shows the physical infrastructure with two geographical areas (equivalent to reservoir) which each contain two subsea templates. Subsea templates (four in total) contain three wells each (twelve in total).

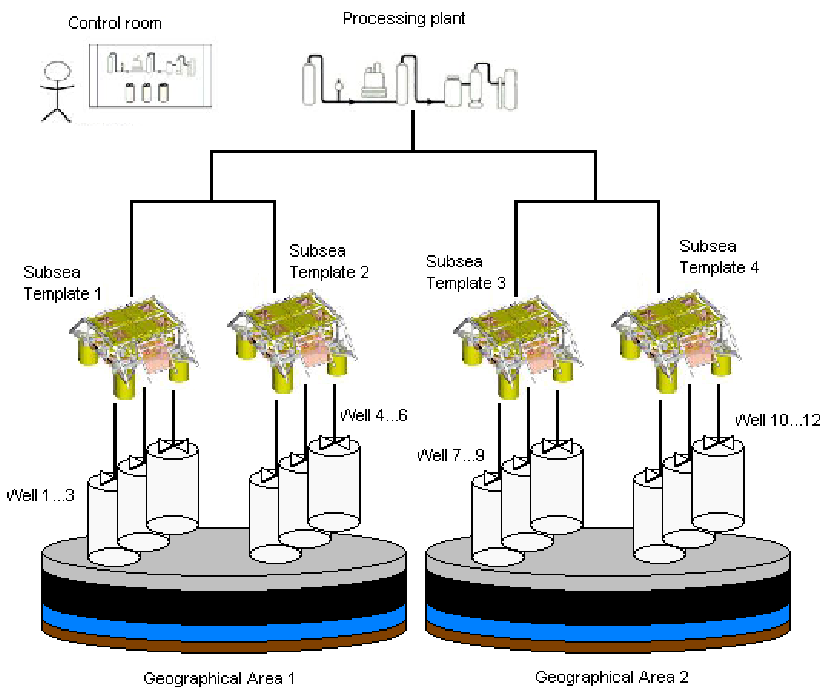


Figure 4: Physical infrastructure of an oilfield

The field has no injection-wells, only production wells. A decrease in oil production due to loss of pressure can therefore not be handled by injection to increase the pressure. To reach the wanted production, causes the production of the different wells to compensate lost production between each other, or the total production target has to be changed according to the current situation.

The field is "process-limited". That is, the production facilities are a potential bottleneck. The wells are also potential bottleneck, called "well-limited". The one of the two limitations preventing the field from reaching its production target varies from time to time. Sometimes it can be the capacity of the processing facilities, and other times it can be that the wells do not deliever enough according to the field production target. The reference problem defined in this chapter assumes "well"-limitation to be the reason if a production target is not achieved.

5.3 Assumptions

This section contains different assumption about the environment and concerns of the system design, that influence how the architecture is built. The assumptions are the following (3):

- A well cannot influence the reservoir pressure and reservoir properties nearby another well. Their influence on each other is assumed to be in little extent, and is therefore neglected within a short period of time. This simplifies the calculation of each well's production, since all well's production forecast calculation is not depending on each other for this period. After this period, the new reservoir pressure and reservoir properties are recalculated and updated.
- Sensor data is cleansed and filtered causing access to valid data.
- The processing plant has no oil capacity limit
- The processing plant has a known fixed gas-, sand- and water capacity.

- The manageable sand limit in the well is known.
- The agent system has access to adjust the chokes.
- The reservoir pressures and properties nearby the wells are known, in order to have the ability to forecast well production.
- Oil is the preferred substance over gas, water, and sand.
- Wells connected to Subsea templates have to have the same wellhead pressure in order to avoid affecting eachother's production.

5.4 Optimal and maximal production

The terms "optimize" and "maximize" are often used in an inseparable manner. Regarding oil production, a distinction must be made between these terms since they are not synonymous (4). Optimization will have as goal to maximize the oil production, the time horizon is different. Optimization is about maximizing the oil production in the long term, taking the extraction of oil in reservoir into consideration. To perform this optimization, long term plans and decision should align, and not maximizing the oil production at the current moment. This alignment can be performed faster than before because of the increased real-time data available, enabling adjustments and forecasts in a more frequently manner.

5.5 Previous experiences

This section describes previous experiences made when developing oil productions systems with agents.

5.5.1 Oil production systems

Ølmheim, Landre & Quale (2) suggested concepts for inclusion in the next generation production support systems. The system is based on use of delegated and variable autonomy. The suggested architecture is shown in Figure 5. It is divided into three distinct layers illustrating the business concerns at each layer.

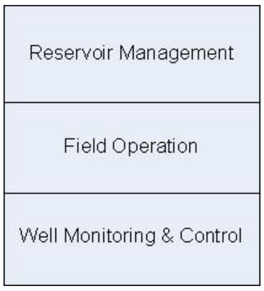


Figure 5: Architectual layers

The layers are described as follows:

- *Reservoir Management:* Responsible for the long term objectives for each reservoir and defines the goals for the lower levels to implement. Uses reservoir models and supports what-if analysis of different scenarios. Determines how each reservoir should be drained for the purpose of maximizing long term value in each reservoir and between reservoirs in a geographical area.

- *Field Operation:* Responsible for the day-to-day operation of a single field. Uses field flowline models and simulations to find the "best possible" configuration. Receives production goals for each of its reservoirs from reservoir management, and develops plans for how to manage the wells based on actual well state and designated production goals.

- *Well Monitoring & Control:* Responsible for monitoring and control of the individual wells. Captures the uniqueness found in each individual well. Compared with more traditional use of layers in software engineering, this approach differs as the components located in each layer will negotiate contracts as part of the delegated autonomy.

Interpreting the algorithms presented in this report, reaching a common goal for a group of agents (teamwork) is here realized through the "commander-agent" of the group knowing what it would like the team-member's contribution to be. The next step is for the "commander-agent" ask all the team-members to make that contribution as best as possible and negotiates to make it work as best as possible. If there exists several wells, negotiation with each of them would cause a large amount of messages going back and forth to settle a production target for the group well, especially if the contribution should be globally- and locally optimized at the same time. If one well cannot meet the planned production target, this can mean renegotiation for the wells who already settle their production target in order to compensate to reach the common production target for the whole group. The system proposed in this report will hav all wells telling the field about their possible contributions and the field (commander of the group of wells) can therefore decide which well shall make what contributions. The focus will be how to reach a common goal (common production target) for the whole group of wells, which implies use of teamwork.

The project preparing for this master thesis (3) designed a system structure that shows a hierarchy with the *Human operators* and *Operator Assistant (OA)* on the top, with the *Optimizing Field Oil Production System (OFOPS)* team underneath (*Plant Monitor* is a sub-team of *OFOPS*), who further delegates the work through the *Subsea Template Collection (STC)* (same as a geographical area mentioned in the system goals), *Subsea Template (ST)*, and finally at the bottom level, the *Well (W)*. The hierarchy is bound together with contracts between the different levels in the hierarchy. In order for the operator to be able to take actions, he/she has to follow the authority lines, so contracts are not broken between teams in an uncontrolled manner.

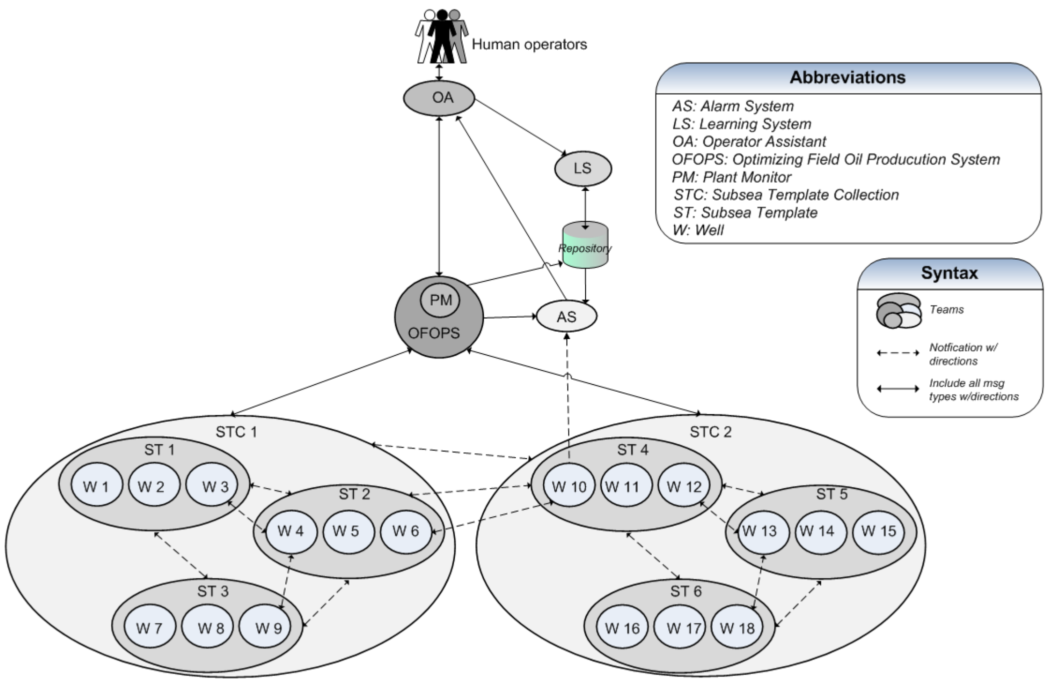


Figure 6: System structure (3)

All teams will have the ability to deal with decisions within their scope of delegated authority. All teams consist of a number of sub-teams which have agreed to work together toward a common goal. The team tries to reach the team goal rather than the local goals of the sub-teams. To support the overall team goal, the sub-teams need to collaborate with each other. The coordination is achieved trough communication, showed in the system structure by arrows between the different teams.

The physical infrastructure in this system structure will be used as a basis for this report. The teamstructure will also be explored and examined. The human operator shall be more directly involved in the different team’s processeses, in a new system. This will for instance possibly result in a relationship of trust between the human operator and the machine system.

Chapter 6

System design

This chapter explains the basic software layers that have to exist in a system that address the reference problem from Chapter 5. The different control processes and "steps of action"-scenarios this type of system is going to handle are described in the following subsections. These elements will generate a core system design that will be the foundation when implementing the JACK Agent version and the JACK Teams version.

6.1 Layers

The suggested architecture is divided into five distinct layers illustrating the encapsulation of functionalities. The layers are shown in Figure 7 in two versions. The ideal architecture has one layer that is placed vertical and is able to interact with all horizontal layers directly. The simplified architecture used in this report has only a user interface towards the "Field Planning & Monitoring"-layer, in order to be able to realize the system within the workhours available in this project.

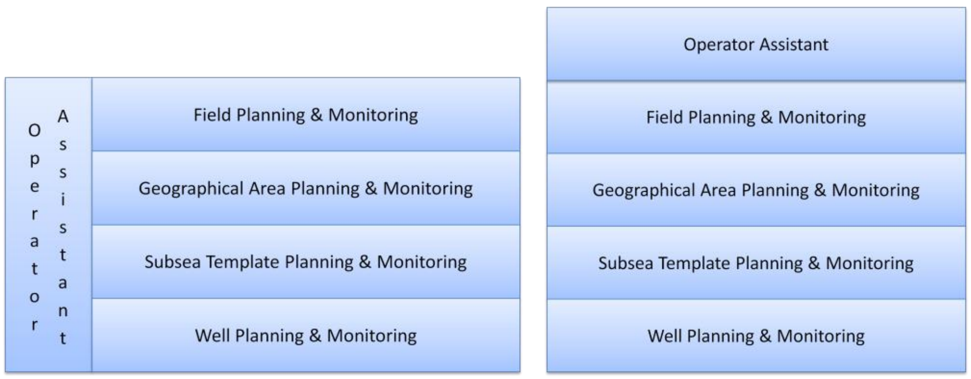

Operator Assistant
Field Planning & Monitoring
Geographical Area Planning & Monitoring
Subsea Template Planning & Monitoring
Well Planning & Monitoring
Operator Assistant
Field Planning & Monitoring
Geographical Area Planning & Monitoring
Subsea Template Planning & Monitoring
Well Planning & Monitoring

Figure 7: Architectual layers – Ideal architecture & Simplified architecture

- ***Operator Assistant:*** This layer is involved in the selection of production-scenarios at the different layers to be able building the field production-scenarios. It is involved in selecting the field production-scenario to plan the production according to, and has the ability to monitor production hour by hour.

- ***Field Planning & Monitoring:*** This layer is responsible for the long term production target, and defines the goals for the geographical area levels to implement. It determines how each geographical area should be drained for the purpose of maximizing long term value of the oil field.

- ***Geographical Area Planning & Monitoring:*** This layer selects the different the production levels for the subsea template connected to each physical geographical area, after consulting with the "Subsea Template Planning & Monitoring"-layer. It coordinates the production levels of the subsea template connected to the geographical area, and captures the information about each individual geographical area.

- ***Subsea Template Planning & Monitoring:*** This layer selects the different the production levels for the wells connected to each physical subsea template, after consulting with the "Well Planning & Monitoring"-layer. It coordinates the production levels of the well connected to the subsea templates, and captures the information about each individual subsea template.
- ***Well Planning & Monitoring:*** This layer contains prediction models and productiondata for each individual well, used in the planning phase and during the production phase. It is responsible for monitoring and control of the individual wells, and captures the information about each individual well.

6.2 Processes

The production system constructed run three processes: *Proactive planning*, *Reactive adjustment*, *Monitoring & Control.* The processes run in a serialized manner, one at a time. The different processes are shown is Figure 8.

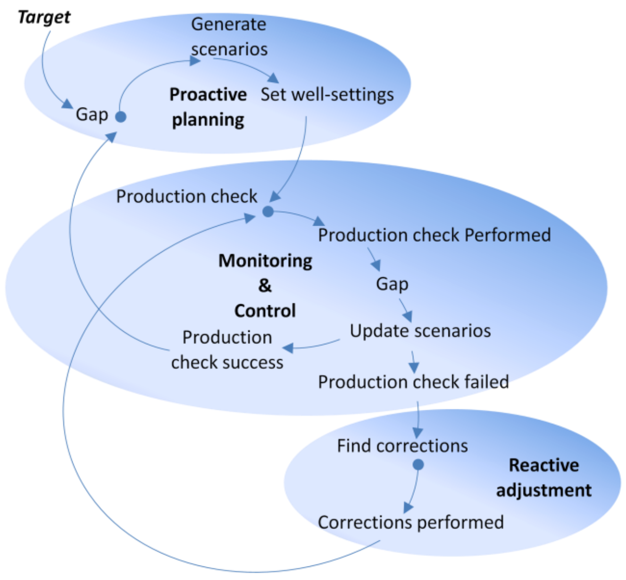


Figure 8: System processes

Proactive planning depends on *Monitoring & Control* to see if the system has reached the production targets, or if it shall switch from *Proactive planning* to the *Reactive adjustment* process.

The *Reactive adjustment* process is dependent on *Monitoring & Control* to see if the system has reached the production targets. Figure 8 shows the system going from *Reactive planning*, and back to *Proactive planning*. This is the ideal interaction between the processes. To simplify the system being built, a transition from *Proactive planning* to *Reactive planning* is the only one being allowed, and not the opposite transition. The following processes are described as follows:

- ***Proactive planning:*** The objective of the *Planning process* is to establish the best possible well choke settings for a period of time and to reach production targets specified by the human operator. This configuration will be challenged when it put into operation due to the dynamics in the real world. Equipment will fail and situations emerge and the *Reactive adjustment process* has to be performed in such case. The generation of possible production targets within a specified period of time is built bottom-up, because each well is telling the subsea template how much it can contribute at different production levels. The subsea template is combining the best combinations (highest oil/waste ratio) of well-contributions, and generates subsea template contribution plans. This repeated on geographical area-level, and at last at the field-level. The field-level uses the human operator-level to communicate with the human being that chooses the total production target for the whole field for the period of time.

- ***Reactive adjustment:*** The objective of reactive adjustment is to establish a best possible configuration for the next hour constrained by a production target and the current situation. Because the production system wants to ensure that the production target is reached, maximizing the oil production at the current moment is performed. This may increase the amount of waste produced in each and every moment while producing. Since a maximizing is performed every moment, the production target will possibly be produced faster, but with a larger load at the equipment during those hours. Well choke settings are tuned after environmental changes have occured, instead of changing the well settings in advance. Depending on how fast a change in the environment is detected in the environment, this could cause some extra time to adjust accordingly. *Reactive adjustment* is top-down, where the field-level asks for the best oil/wast-ratio subsea templates, then adjusting and selecting which subsea template to produce at what levels. This is done at subsea template-level because all wells connected to a subsea template have to have the same well-head pressure to avoid producing into one another. This causes the all the wells connected to it to run with the same well choke settings.
- ***Monitoring & Control:*** The production is being monitored and controlled after each time step in both the *Proactive planning* process and the *Reactive adjustment* process. The *Monitoring & Control* process decides the system state (proactive or reactive) for the next hour. Note that the system design is simplified and can only change from the proactive state to the reactive state, and not the opposite.

6.3 System operation

This section describes how the system implements global- and local optimization and steps of action"-scenarios the system shall be able to do.

6.3.1 Global optimization

The system processes presented in Figure 8 will be described using "steps of action"-scenarios, where one "steps of action"-scenario describes normal production (equal to the *Proactive planning* process) and the second "steps of action"-scenario describes how to deal with unexpected change (equal to the *Reactive adjustments* process).

The system emphasizes the use of global optimization, while still having every component producing within its limits and restrictions. Global optimization will cause production levels with the highest oil/waste ratio in total and with the lowest total amount of waste as possible, which is important for the production facilities in order to avoid using unnecessary resources and capacities.

The production system constructed in this report asks every well component in the lowest level in the physical infrastructure hierarcy shown in reference problem presented in Chapter 5. They report back to the subsea template they are connected to, telling what amounts of oil and waste they are able to produce (within a zone of local optimization). This process is repeated at the different levels until reaching the field-level in the hierarchy. A global optimization for the whole field will take place at this moment. At every level in the hierarchy a few production-scenarios are removed because of local optimization. Within this zone, the globally optimized production-scenario is chosen. Every possible combination of well settings is generated because of the process of building a globally optimized set of well production bottom-up. The bottom-up approach performed in this project will create a globally optimized set of well settings, while a top-bottom approach would have created approximately globally optimized and make the lower levels do local optimizations.

All processes in the system are initiated at the top level. Figure 9 shows how the initiation of actions starts on the top level and propagates downwards, while returning to the top level after reaching the bottom level. Reaching the top level, a globally optimized decision is ready to be made.

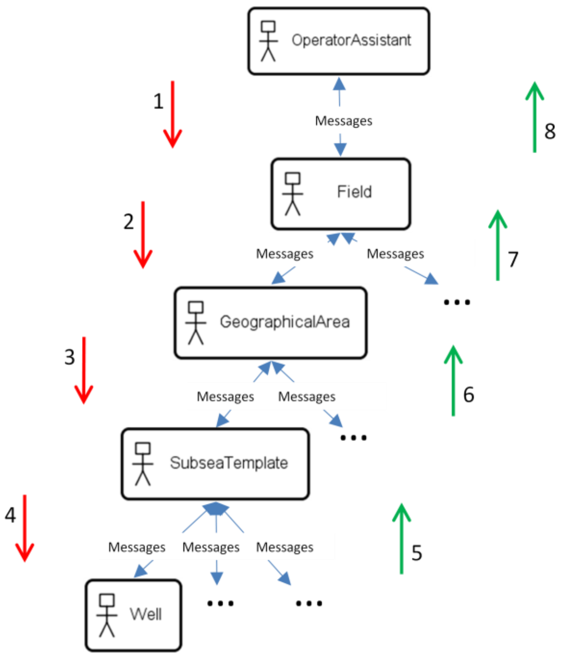


Figure 9: Path of information flow

6.3.2 “Steps of action”-scenarios

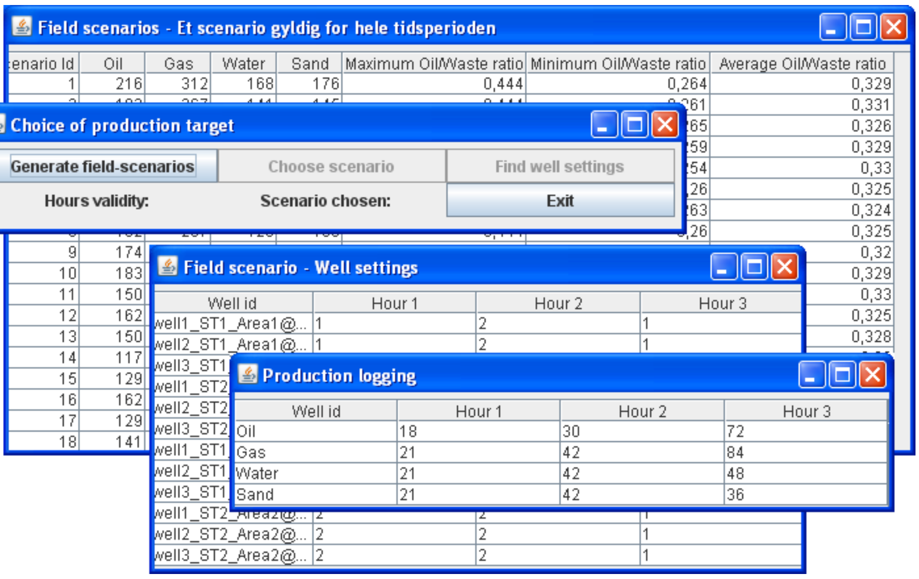
The “steps of action”-scenarios described in this section represent the work that the implemented system shall do during runtime are. The system can be represented by two main “steps of action”-scenarios. The pseudo-algoritms realizing these “steps of action”-scenarios can be found in Appendix A. The two “steps of action”-scenarios are described as follows:

- ***“Steps of action”-scenario one - Planned and predicted production:*** The “steps of action”-scenario involves the *Proactive planning*- and the *Monitoring & Control* control processes. The scenario starts when the length of the period of time is selected. Knowing the length on the period of time, different production-scenarios with different production targets is generated. The selected production target for the period of time is the production target selected from the predicted field production-scenarios. Selecting a predicted production-scenario is performed by a human operator defining the period of time he/she wants to produce within. The objective is to achieve the amount of oil and waste (defined as gas, water, and sand in this project) specified in the production-scenerio, which is done by following the planned well choke settings hour by hour.

- ***"Steps of action"-scenario two - Unpredicted changes according to planned production:*** The "steps of action"scenario involves the *Reactive adjustment*- and the *Monitoring & Control* control processes. The scenario starts when the predicted field production-scenario does not fit the actual production. The objective now is to deal with the unpredictability of the environment, while optimizing globally at the next timestep. The prime goal is to reach the production target within acceptable production levels, but may cause intensive production (same amount of production, but in less time) that could cause a greater extent of corrosion on equipment, and increase the risk of technical failures. The process is repeated hour by by hour to the end of the period of time selected, or until the production targets have been reached.

6.4 Simulated environment

The graphical user interface (GUI) used in this project is shown in Figur 10. It is used to generate proactive plans with different possible production targets, during a specified period of time. Another feature is to look at the actual well choke settings that the agent system will use when following a specific plan. The agent system will follow the proactive production plan set for the production-scenario when the human operator selects a production-scenario. A change to reactive state will be made if the actual production does not follow this plan. This is not shown in the GUI, but is performed in the automatically in machine system.



Field scenarios - Et scenario gyldig for hele tidsperioden

enario Id	Oil	Gas	Water	Sand	Maximum Oil/Waste ratio	Minimum Oil/Waste ratio	Average Oil/Waste ratio
1	216	312	168	176	0,444	0,264	0,329
						61	0,331
						65	0,326
						59	0,329
						54	0,33
						26	0,325
						63	0,324
						26	0,325
9	174						0,32
10	183						0,329
11	150						0,33
12	162						0,325
13	150						0,328
14	117						
15	129						
16	162						
17	129						
18	141						

Field scenario - Well settings

Well id	Hour 1	Hour 2	Hour 3
well1_ST1_Area1@...	1	2	1
well2_ST1_Area1@...	1	2	1
well3_ST1			
well1_ST2			
well2_ST2			
well3_ST2			
well1_ST1			
well2_ST1			
well3_ST1			
well1_ST2			
well2_ST2_Area2@...	2	2	1
well3_ST2_Area2@...	2	2	1

Production logging

Well id	Hour 1	Hour 2	Hour 3
Oil	18	30	72
Gas	21	42	84
Water	21	42	48
Sand	21	42	36

Figur 10: Graphical User Interface

The environmental variables are predicted oil-, gas-, water-, and sand production/hour in each well. The numbers are contained in a textdocument that draw an interface towards calculations of proactive production-plans. Production data to monitor is also given by a textdocument, to represent the sensor values hour by hour. The values are not selected randomly, because this it is irrelevant to the experiment conducted.

6.5 Applied teamwork and implications

The system designed in this chapter has teamwork at field-, geographical area-, and subsea template level. The common teamgoals for the teams was made out from a combination of what amount of production the teammembers can offer during the period of time or the next hour. The field team is the only team having the same teamgoal for the whole period of time selected and the possibility to change the system state. The other teams have a common teamgoal for the whole period of time if in proactive state, and a common teamgoal substituted every hour if the system is in a reactive state.

Subsection 2.1 indicates three challenges regarding multi-agent systems: there exists *no global system control*, *data is decentralized*, and that the *computation is asynchronous*. Teamwork was pointed out a possible solution addressed to these challenges. The system designed contains *global system control* due to the hierarchy built. The *data being desentralized* challenge is not an issue here since the data that is desentralized on different hierarchical levels belongs to different abstraction levels. *Asynchronous computation* creates computation at different abstraction levels. Different abstraction levels will create a good decomposition and less calculations necessary, in order to create a global optimal solution.

Many approaches exist in the work of coordination of agent systems. Subsection 2.5.2 described several coordination techniques that can be used in a multi-agent system. *Contracting* is the coordination technique used to realize the teamwork in the designed oil production system of this report is named. First the team announces a contract, receives bids from its teammembers, evaluates the bids, and finally awards the different contracts to the subteams. Analogous the team announces a contract of produciton for a specified period of time, receives production-scenarios from its teammembers, evaluates the production-scenarios, and finally awards the different contracts (id of the production-scenario to follow) to the subteams. Awarding contracts (id of the production-scenario to follow) happens after after the finding of the best composition of all production-scenarios received.

Subsection 2.5.3 went through team variables. The different team variables are addressed in this design as follows:

- *Team size:* Team size is chosen on background of the reference problem from Chapter 5. One field team contains two geographical areas subteams. A geographical Area team contains two subsea template subteams (totally four in the whole system structure). A subsea template team contains three well subteams (totally twelve in the whole system structure).

- *Team structure:* The teamstructure is static during runtime due to the static relationship the physical components has to eachother in the reference problem defined. Solving a teamtask, all the subteams will be given the same type of sub-tasks since they behave in the exact same way on the different hierarchical levels. Since the subteams represent different physical components, the tasks will be solved according to that unique instance of the physical component.

- *Differentiation and Specialization:* The teams are homogenous. This causes teammembers to take over and fulfil a role for a teammember that is temporary out of function, if all beliefs needed are available. This was however not included in the system design.

- *Failure handling:* The system is designed with a fixed step of actions, with strict control exercised from the team on the highest hierarchical level (field team). If a well cannot produce according to its contract it will have consequences for the whole system, because the system tries to achieve a globally optimal production solution. The field team will initiate a check everyhour to examine all contracts in the system, to see if they still are sustained. If a contract is violated, a violation report will propagate to the field team that will change the system state from proactive to reactive and initiate an adjustment.

- *Authority delegation:* The team on the highest hierarchical level (field team) initate all control processes in the system. This creates a global control, which delegate subteams to filtrated and monitor information on lower abstraction levels. The human operator has delegated to the machine system everything except choosing the length on the period of time to produce, the selection of field production-scenario, and production monitoring.

6.6 Towards a human-centric system

Human operator is able to influence the machine system in the planning phase of the oil production, as described in Section 6.5. Interference from the human operator is not possible from the human operator when actual production has started. Some of the human-centric principles from Subsection 2.4.3 have been used as background for this involvement of human operator. This is not the mainfocus of the report and is not prioritized. The main focus is to compare JACK Agents and JACK Teams and to evaluate their applicability and suitability in teamwork construction.

Chapter 7

JACK Agents solution

This chapter runs through the design and implementation used to construct the JACK Agent version of the program to be developt. The design shows the system structure of all the agents that will be instanciated, and how the different kinds of agents realize teamwork. How the teamwork is done in the actual implementation, is looked into by studying the primitives used by agents and plans, and how they function.

7.1 System structure

The system structure of agent-instances used in the particular program developt is shown in Figure 11. The structure consists of 1 operator assistant, 1 field, 2 geographical areas, 4 subsea templates, and 12 wells. The numbers of agents are the same as physical components given in the reference problem described in Chapter 5. The interaction follows the same path as in a military structure. All commandoes and enquiries have to follow a hierarchical path. The different levels in the hierarchy represent different levels of abstraction levels. This will eventually lead to global optimization and put the operator assistant in control and represent information on a suitable level of abstraction.

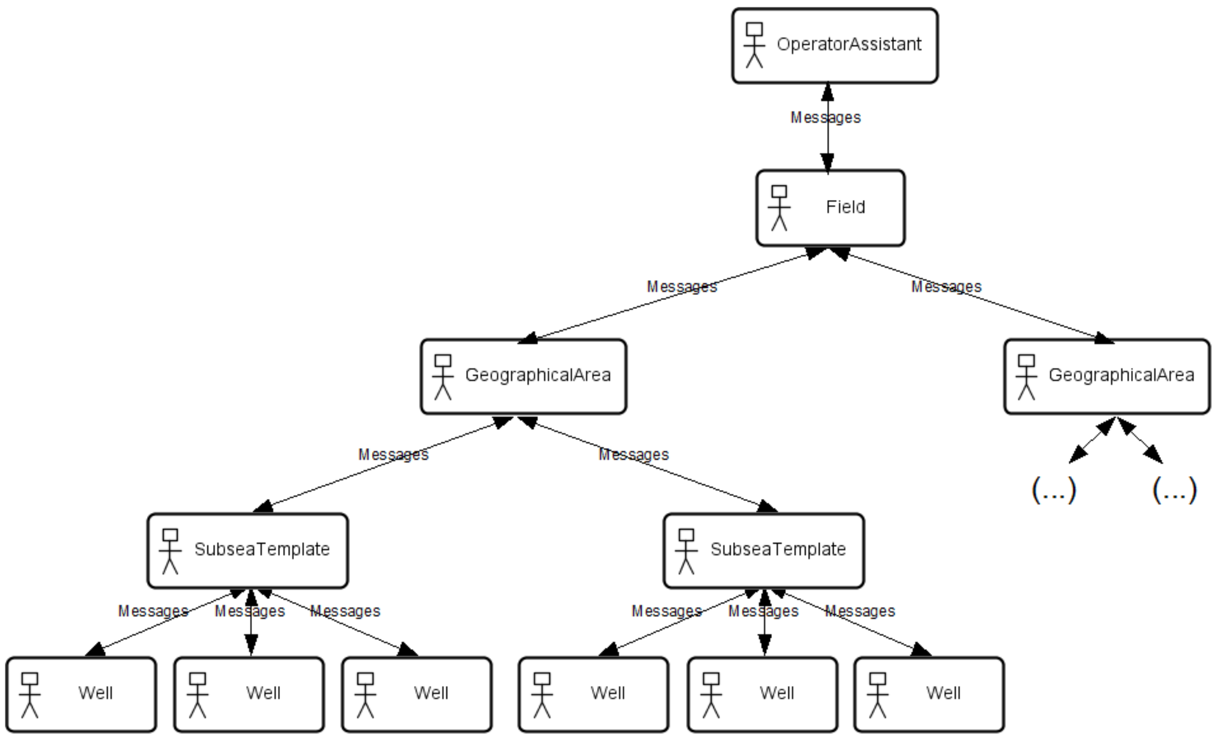


Figure 11: System structure - JACK Agents solution

The conceptual/intended teamstructure in the JACK Agent solution is as following starting with the smallest teams:

- The *Subsea Template team* has Wells as teammembers.
- The *Geographical Area team* has Subsea Templates as teammembers.

- The *Field team* has Geographical Areas as teammembers.
- The operator assistant is involved in the teamprocesses regarding the *Field team*.

7.2 Teamwork

Two "steps of action"-scenarios were defined in Subsection 6.3.2: "*Planned and predicted production*" and "*Unpredicted changes according to planned production*". These scenarios represent the work that the system does. The same "steps of action"-scenarios are implemented in the JACK Teams solution, and create a foundation that makes the two versions suitable for comparison. The parts that will be different in the two versions are the subsea template- and well levels, and is therefore the main-focus in this section.

The two "steps of action"-scenario are divided into several interaction sequences to easier see what happens during the scenario. The *"Generate production-scenarios"-scenario* is part of the "*Planned and predicted production"-scenario*, and is representative what concerns teamwork. This section will therefore describe it in details. For more information about the other interaction sequences, see Appendix B.

The *"Generate production-scenarios"*-scenario will show how teamwork was built in JACK Agents. The scenario begins when the proactive planning process receives a period of time, for which different production-scenarios with different production targets is fulfilled. Figure 12 shows all agents, plans, and events involved in this process.

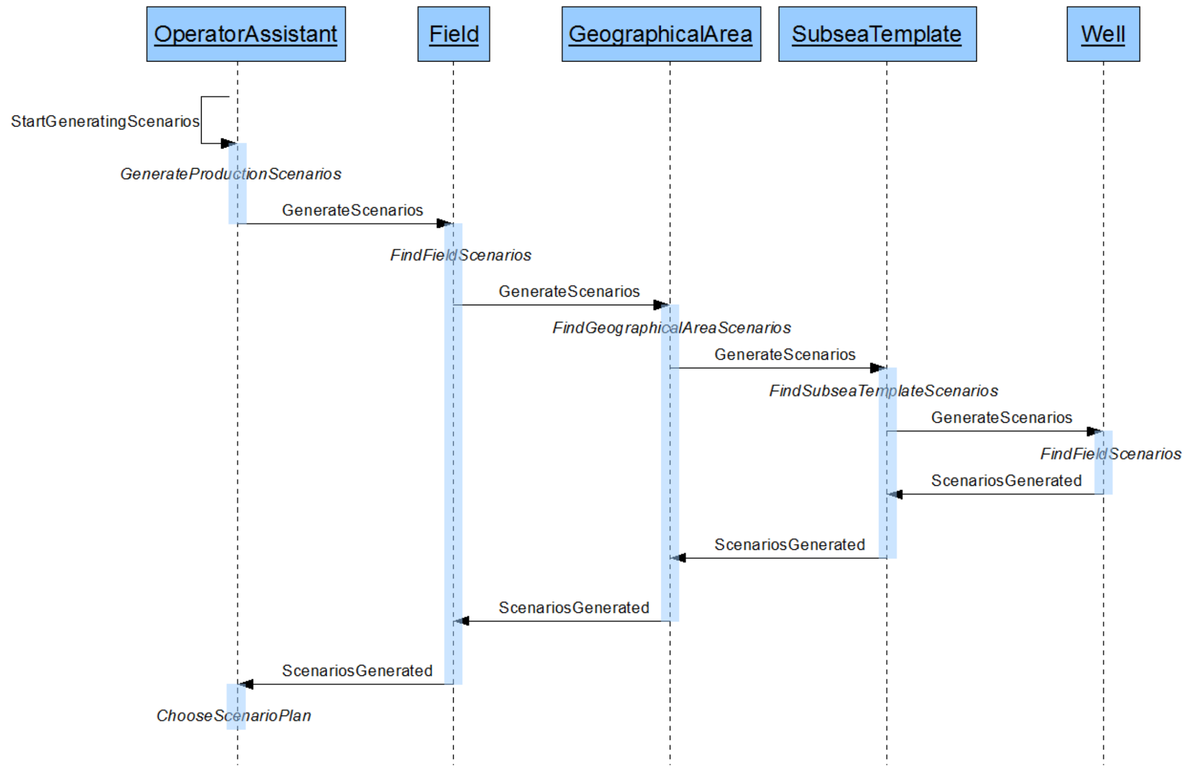


Figure 12: Generate production-scenarios

The human operator has to specify a valid period of time through the operator assistant agent. The number of hours is propagated downwards the system. When the event containing the number of hours reaches the hierarchical bottom-level well, well production-scenarios are generated.

Propagating to the the subsea template team (with well as teammember), well production-scenarios are combined into subsea template production-scenarios. The same process is repeated for the geographical area team, with subsea template production-scenarios. The final repetition of this process generates field production-scenarios, being ready to be choosed among with different production targets.

The implementation of the subsea template teamwork is shown in Figure 13. To reach the well teammembers, a list containing the addresses to all connected well teammembers are accessed. This list was established in the main Java-method.

Communication is implemented by using the @send and @reply statements. Cooperation is getting all teammembers to create the best subsea template production-scenarios. They have to use the same well choke settings at all times. Coordination is that all well teammembers asked each time the subsea template is asked for subsea template production-scenarios. The well teammembers is asked in a serial fashion, but this could have been done in parallel as well. The common team goal in this proactive state is the different subsea template production-scenarios' production goals hour by hour. One of these subsea template goals has to be selected by the geographical area.

```
public plan FindSubseaTemplateScenarios extends Plan {
(...)

#reasoning method
body()
{
        (...)
        for(int i=0; i<connectedWells.size(); i++) {
                GenerateScenarios q = ev2.generateWellScenarios(hoursValidity);
                @send( (String)connectedWells.get(i), q );
                @waitFor( q.replied() );
                ScenariosGenerated wev = (ScenariosGenerated) q.getReply();
                Scenarios wellScenarios = wev.scenarios;
                generatedWellScenarios.add(wellScenarios);
        }//end for-loop
        subseaTemplateScenarios = generateSubseaTemplateScenarios(generatedWellScenarios);
        self.setSubseaTemplateScenariosList(subseaTemplateScenarios);
        @reply(ev, ev1.scenariosGenerated(subseaTemplateScenarios) );
}//end body()

(...)
}//end plan
```

Figure 13: Subsea Template - FindSubseaTemplateScenarios-plan

```
public plan FindWellScenarios extends Plan {
(...)

#reasoning method
body()
{
        wellScenarios = findWellScenarios();
        @reply(ev, ev1.scenariosGenerated(wellScenarios) );
}//end body()

(...)
}//end plan
```

Figure 14: Well - FindWellScenarios-plan

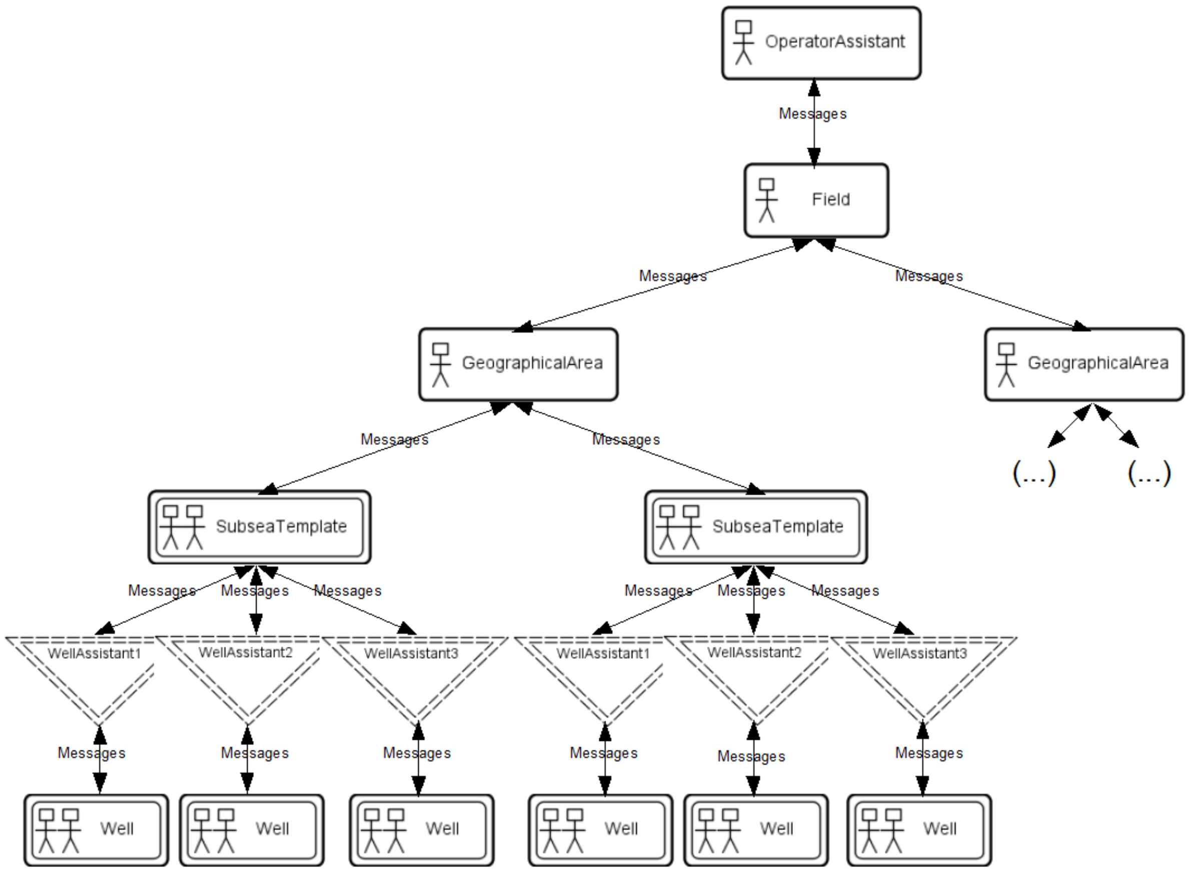
Chapter 8

JACK Teams solution

This chapter runs through the design and implementation used to construct the JACK Teams version of the program to be developt. This version has changes made in the subsea template and well compared to the JACK Agents solution in Chapter 7. The design shows the system structure of all the agents and teams that will be instanciated, and how the different kinds of agent realize teamwork. How the teamwork is done in the actual implementation, is looked into by studying the primitives used by agents and plans, and how they work. The structure and elements in this chapter is the same as the design used in the JACK Agents solution.

8.1 System structure

This section shows the system structure used in the JACK Teams solution. Figur 15 shows the system structure consisting of 1 operator assistant, 1 field, 2 geographical areas, 4 subsea templates, and 12 wells. The numbers of agents/teams are set according to the reference problem in Chapter 5. The hierarchy is build the same way as the JACK Agents solution and communication has to follow the communicationpaths it implies. Each level deals with a different abstraction level, and the top-level agent handles the initiation of the different sequences of actions described through the "steps of action"-scenarios in Section 6.3.2. All initiation of action will therefore lead global optimization dealing with the highest abstraction level of information composite from the abstraction levels lower in the hierarchy.



Figur 15: System structure - JACK Teams solution

Some of the teamwork is represented by using agent-instances from JACK Agents to form teams, and the subsea template and wells are using the team-construct from JACK Teams. The teamstructures in the JACK Teams version are as follows:

- The *Subsea Template team* (using the team-construct) has Wells as teammembers (using the team-construct).
- The *Geographical Area team* (using the agent-construct) has Subsea Templates as teammembers (using the team-construct).
- The *Field team* (using the agent-construct) has Geographical Areas as teammembers (using the agent-construct).
- The operator assistant (using the agent-construct) is involved in the teamprocesses regarding the *Field team* (using the agent-construct).

8.2 Teamwork

Two "steps of action"-scenarios were defined in Subsection 6.3.2: "*Planned and predicted production"* and "*Unpredicted changes according to planned production".* These scenarios represent the work that the system does. The same "steps of action"-scenarios are implemented in the JACK Agents solution, and create a foundation that makes the two versions suitable for comparison. The parts that will be different in the two versions are the subsea template- and well levels, and is therefore the main-focus in this section.

The two "steps of action"-scenario are divided into several interaction sequences to easier see what happens during the scenario. The *"Generate production-scenarios"-scenario* is part of the "*Planned and predicted production"-scenario*, and is representative what concerns teamwork. This section will therefore describe it in details. For more information about the other interaction sequences, see Appendix B.

The *"Generate production- scenarios"*-scenario will show how teamwork was built in JACK Teams, and will be described in details in this section. The scenario begins when the proactive planning process receives a period of time, for which different production-scenarios with different production targets is fulfilled. Figure 16 shows all teams, agents, plans, and events involved in this process.

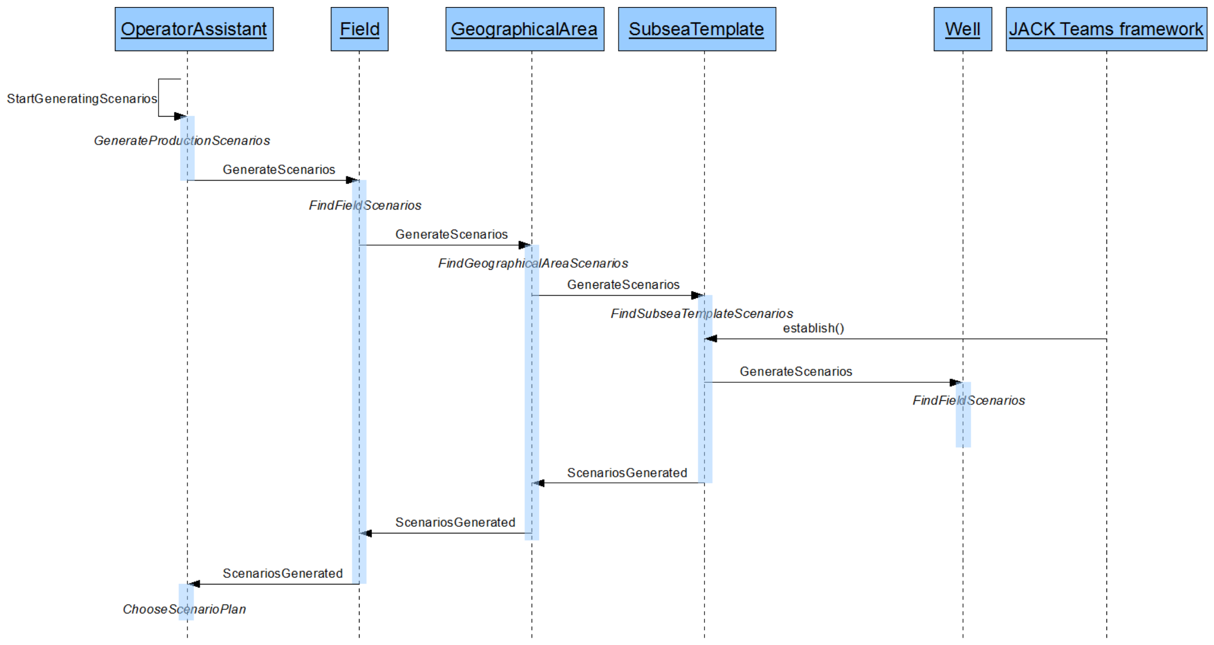


Figure 16: Generate production-scenarios

The scenario starts with the human operator specifying a valid period of time through the operator assistant agent. The number of hours is propagated downwards the system. When the event containing number of hours reaches the hierarchical bottom-level well, well production-scenarios are generated. Propagating to the the subsea template team (with well as teammember), well production-scenarios are combined into subsea template production-scenarios. The same process is done for the geographical area team, with subsea template production-scenarios. The final repetition of this process generates field production-scenarios, being ready to be choosed among with different production targets.

Figure 17 shows the teamwork for the subsea template team consisting of well subteams as teammembers. The implementation of the well's plan is shown in Figure 18. Communication is implemented by using the the event sent from subsea template using the @teamAchive statement. The well will change an attribute on the event sent, by using the setScenarios()-method (user defined method) on the event.

The @teamAchieve statement will maintain the event it sent for any updated attributes after the subsea template has succeded to respond to the event. Cooperation is having the teammembers (wells) to work towards the common team goal. The common team goal is to have the subsea template production-scenario to produce every hour with the production-scenarios inititially selected by the field. To reach this goal a team-subgoal is generate subsea template production-scenarios performed during these steps of actions. Coordination is having the well teammembers asked in a serial fashion what amount of oil and waste the can produce, but this could have been done in parallel as well. The subteams fulfilling the required roles needed by the team are fulfilled by the establish()-method used by the framework. This establish()-method can been seen in Figure 17.

```
public teamplan FindSubseaTemplateScenarios extends TeamPlan {
(...)

#reasoning method
establish()
{
        Vector busy = new Vector();
        wellAss1= (WellAssistant1) pickRole( busy, wa1 );
        wellAss1!= null;
        wellAss2= (WellAssistant2) pickRole( busy, wa2 );
        wellAss2!= null;
        wellAss3= (WellAssistant3) pickRole( busy, wa3 );
        wellAss3!= null;
}

#reasoning method
body()
{
        (...)
        find(wellAss1);
        find(wellAss2);
        find(wellAss3);
        (...)
}//end body()

#reasoning method
find(Role wellAss)
{
        (...)
        if(wellAss instanceof WellAssistant1) {
                WellAssistant1 wellAssCasted = (WellAssistant1)wellAss;
                GenerateScenarios q = wellAssCasted.gs.generateWellScenarios(hoursValidity);
                @teamAchieve( wellAssCasted, q);
                wellScenarios = q.scenarios;
        }
        else if(wellAss instanceof WellAssistant2) {
                WellAssistant2 wellAssCasted = (WellAssistant2)wellAss;
                GenerateScenarios q = wellAssCasted.gs.generateWellScenarios(hoursValidity);
                @teamAchieve( wellAssCasted, q);
                wellScenarios = q.scenarios;
        }
        else if(wellAss instanceof WellAssistant3) {
                WellAssistant3 wellAssCasted = (WellAssistant3)wellAss;
                GenerateScenarios q = wellAssCasted.gs.generateWellScenarios(hoursValidity);
                @teamAchieve( wellAssCasted, q);
                wellScenarios = q.scenarios;
        }
        else {
                (...)
        }
        (...)
}//end find()

(...)
}//end plan
```

Figure 17: SubseaTemplate - FindSubseaTemplateScenarios-plan

```
public teamplan FindWellScenarios extends TeamPlan {
(...)

#reasoning method
body()
{
        wellScenarios = findWellScenarios();
        ev.setScenarios(wellScenarios);
}//end body()

(...)
}//end plan
```

Figure 18: Well - FindWellScenarios-plan

Part III

Results and conclusions

Chapter 9

Quantitative results

This chapter contains results from the quantitative part of the experiment documented in this report. The JACK Agents solution (see Chapter 7) and the JACK Teams solution (see Chapter 8) are evalutated according to the metrics defined experiment. Metrics from Chapter 4 are used to make measurements that either confirm or reject hypotheses formulated in the experiment. The validity of the result is discussed at the end of the chapter, where some of the validity threats have been addressed, while others have been accepted.

9.1 Testing of hypotheses

The measurements made in the experiment are done according to the metrics presented in Subsection 4.1.5. These metrics have the purpose of getting measurements that are needed test the hypotheses. The hypotheses described in the following subsections are confirmed or rejected in accordance to measurements shown followed by a discussion.

9.1.1 Hypothesis 1

The hypothesis

Hypothesis 1 is concerned with lines of code and is given as follows:

- *H01:* The functionality of the the two versions will be implemented with approximately the same number of code lines.
- *HA1.1:* The JACK Teams version will implement the same functionality as JACK Agents version with fewer lines of code.
- *HA1.2:* The JACK Teams version will implement the same functionality as JACK Agents version with more lines of code.

Measurements

Metric *M1 Lines of Code (LOC)* is used in the testing of hypothesis 1. This metric represent the total number of code-lines in the code written by the system developer. All entities in the different packages will be counted. The packages that are included in the counting process are only the packages that contain differences in JACK Agents solution and the JACK Teams solution. The packages excluded are: Operator assistant, Field, Geographical area, and the GUI package. The measurements of the metric will be found by counting the number of semicolons using Microsoft Word's counting function. The results can be found in Table 4. The result is presented in a diagram in Figure 19 that illustrates the measurements related to each version.

M1: Lines of Code (LOC)

Package	*JACK Agents version*	*JACK Teams version*
Operator assistant	-	-
Field	-	-
Geographical area	-	-
Subsea template	453	582
Well	448	429
GUI	-	-
Scenario structures	156	156
System events	10	13
Main-method	108	130
Total Lines of Code	*1175*	*1310*

Table 4: Results for M1: Lines of Code (LOC)

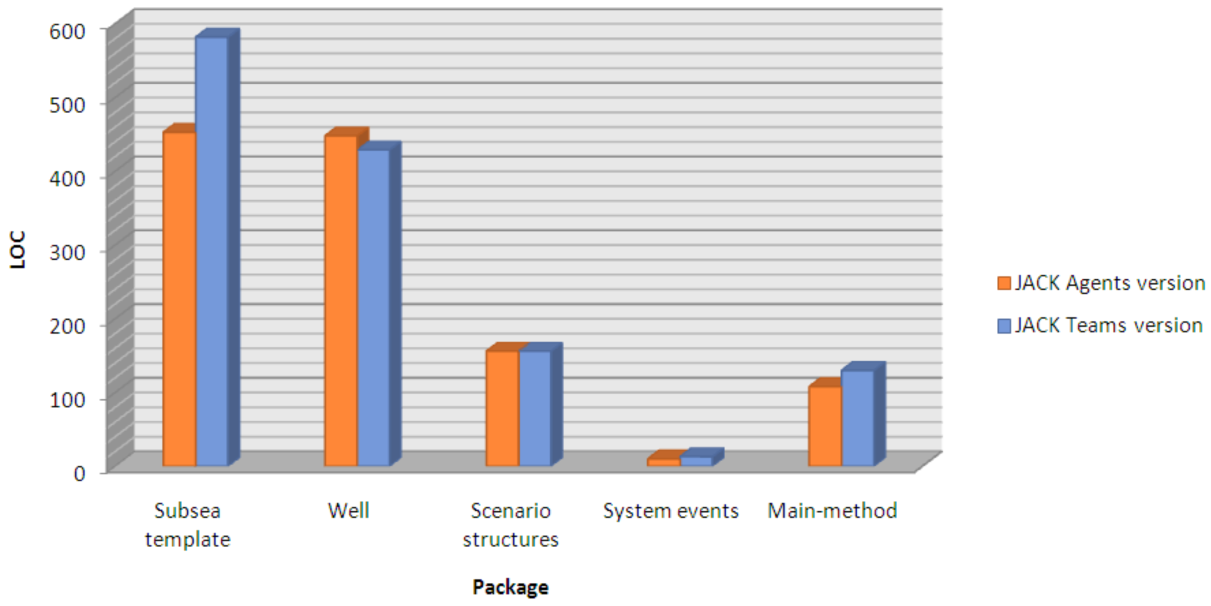


Figure 19: Measurements of Metric M1 (LOC)

Discussion

The diagram in Figure 19 shows that the subsea template package varies a lot between the two solutions. The difference is caused by the extra code needed to establish teams in the JACK Teams solution, while the JACK Agents solution has more reuse of code. The JACK Agents solution use for-loops and repeat the work that needs to be done for each connected teammember. The JACK Teams solution does not use a for-loop, but specify work to each teammember in separate codelines. Using a for-loop was found difficult since the object will not be of the same type, because each well-role is a separate Java-class.

The system events package had some attributes added because of the data exchange by using @teamAchieve in the JACK Teams solution. The main-method package had some extra code-lines because of a bug in the framework, and the instances had to be checked if they were finished being instanciating before moving on. The connection between team and teammembers is not needed in the main-method package to build the structure, since the JACK Teams version uses a .def-file instead to build the possible team structures. The JACK Teams version's SetSubseaTemplateScenarioChosen-plan (from the subsea template package) registrered its teammembers too. This was not done because it needed the addresses to find the teammembers, but it was necessary to be able to find the right production-scenarios for the right teammember.

The JACK Agents version has 1175 lines of code to realize the teamwork, while the JACK Teams version has 1310 lines of code. This is an increase of 11.49%.

Conclusion

The results from the measurements of metric M1 shown in Table 4 indicate that the JACK Teams version has more lines of code than the JACK Agents. Hypothesis H01 is therefore rejected and the alternative hypothesis HA1.2 is chosen.

9.1.2 Hypothesis 2

The hypothesis

Hypothesis 2 concerns the number of entities in each version and is given as follows:

- *H02:* The number of entities will be the same for the two oil production system versions.
- *HA2.1:* The JACK Teams version will have more entities than JACK Agents version.
- *HA2.2:* The JACK Teams version will have fewer entities than JACK Agents version.

Measurements

Metric *M2 Number of Entities (NOE)* is used in the testing of hypothesis 2. This metric represent the number of JACK entities. In the JACK Agent solution this will be the events, plans, capabilities, agents, views, beliefset, and Java classes. The additional constructs in JACK Teams are teamplans, roles, and teams. Plans and teamplans, and agents and teams will fall in the same category. Roles will be a separate entity category. The measurements of the metric will be found by counting the number of JACK entities. The result of the countingprocess is found in Table 5. The result is presented in a diagram in Figure 20 that illustrates the measurements related to each version.

M2: Number of Enities (NOE)

Entity	*Package*	*JACK Agents version*	*JACK Teams version*
Events	Operator assistant	2	2
	Field	8	8
	Geographical area	7	7
	Subsea template	7	7
	Well	3	0
	System events	2	2
Total number:		*29*	*26*
Plan/teamplan	Operator assistant	3	3
	Field	5	5
	Geographical area	5	5
	Subsea template	5	5
	Well	5	5
Total number:		*23*	*23*
Role	Operator assistant	-	0
	Field	-	0
	Geographical area	-	0
	Subsea template	-	0
	Well	-	3
Total number:		*0*	*3*
Capabilities		-	-
Total number:		*0*	*0*
Agents/Teams	Operator assistant	1	1
	Field	1	1
	Geographical area	1	1
	Subsea template	1	1
	Well	1	1
Total number:		*5*	*5*
Views		-	-
Total number:		*0*	*0*
Beliefset		-	-
Total number:		*0*	*0*
Java classes	GUI	7	7
	Main-method	1	1
	Scenario structures	4	4
	Subsea template	2	2
	Well	1	1
Total number:		*15*	*15*
Total Number of entities		***72***	***72***

Table 5: Results for M2: Number of Entities (NOE)

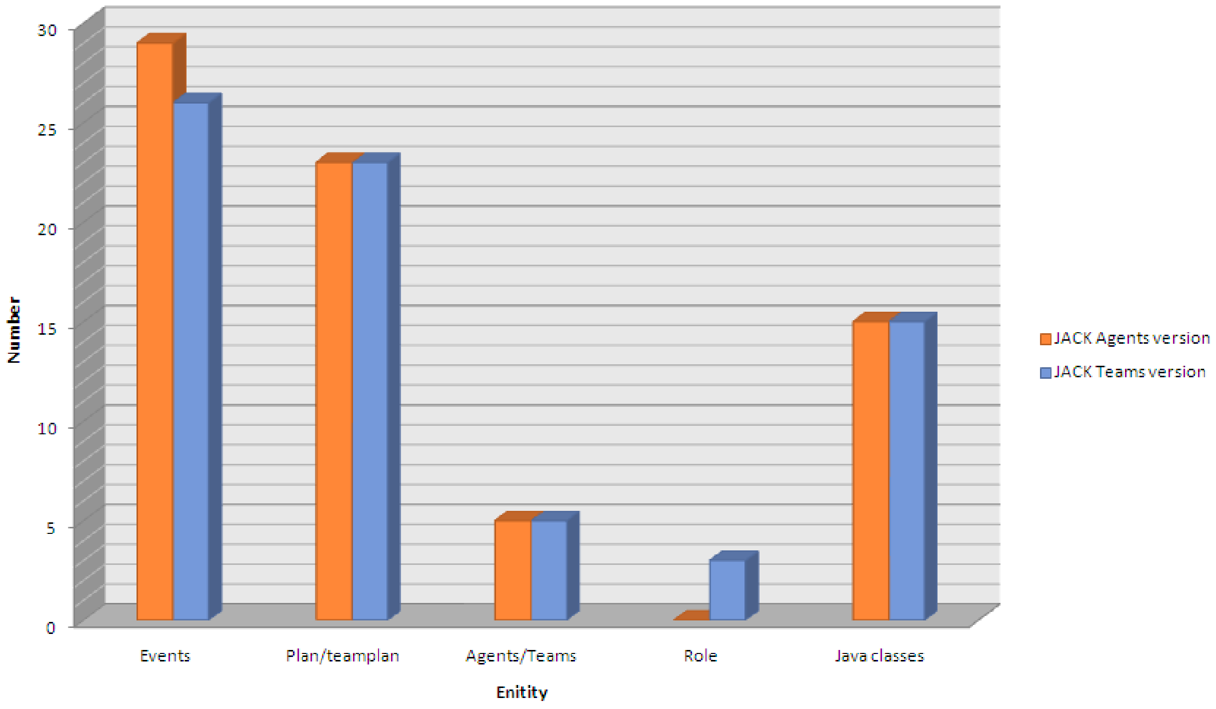


Figure 20: Measurements of Metric M2 (NOE)

Discussion

The differences between the two solutions lays the event package and in the role package, as shown in Figure 20. The JACK Teams solution has fewer events than the JACK Agents solution. This is caused by the data exchange through the @teamAchieve statement, instead of using an extra event sent by the @reply statement as in the JACK Agents solution. The role package is only used by the JACK Teams version.

The JACK Agents version has 72 entities to realize the teamwork, while the JACK Teams version has also 72 entities. The difference in the number of entities between the two versions is therefore 0.0%. This is caused by the number of events and roles neutralizing eachother in this experiment. Increased data exchange using the same number of roles, will however decrease the number of event-entities in the JACK Teams version compared to the JACK Agents version.

Conclusion

The result from the measurements of metric M2 is shown in Table 5, and indicates that the number of entities will be the same for the two solutions. Adding more teammembers to a team, which functions together at the same time without data exchange, was experienced during developement phase to cause that HA2.1 would be the chosen hypothesis. Increased data exchange between team and teammembers was experienced during developement phase to cause that hypothesis HA2.2 would be the chosen hypothesis. There is no proven or unproven proportional dependency between event-entities and role-entities in this experiment, and their unproven dependency (in any) makes it hard to generalize about the results. The null hypothesis H02 is chosen since the number of entities is the same in both solutions used in this experiment.

9.1.3 Hypothesis 3

The hypothesis

Hypothesis 3 concerns the number of functions in each version and is given as follows:

- *H03:* Both versions will use the same number of functions to complete the designed oil production system given in Chapter 6.
- *HA3.1:* The JACK Teams version will complete the designed oil production system (see Chapter 6) with fewer functions than JACK Agents version.
- *HA3.2:* The JACK Teams version will complete the designed oil production system (see Chapter 6) with a larger number of functions than JACK Agents version.

Measurements

Metric *M3 Number of Functions (NOF)* is used in the testing of hypothesis 3. This metric represent the total number of JACK-methods, Java-metods, and plans/teamplans used by the agents, teams, and their own plans/teamplans. The JACK methods are methods defined specially by the framework in the agents/teams and plans/teamplans. The Java-methods are the number of user defined Java-methods in the agent/teams and plans/teamplans. The packages looked into are the subsea template package and well package that realize teamwork differently in the two solutions. The results can be found in Table 6. The result is presented in a diagram in Figure 21 that illustrates the measurements related to each version.

M3: Number of Functions (NOF)

Functions	*JACK Agents version*	*JACK Teams version*
JACK-methods	30	40
Java-methods	47	41
Plans	10	10
Total Lines of Functions	*87*	*91*

Table 6: Results for M3: Number of Functions (NOF)

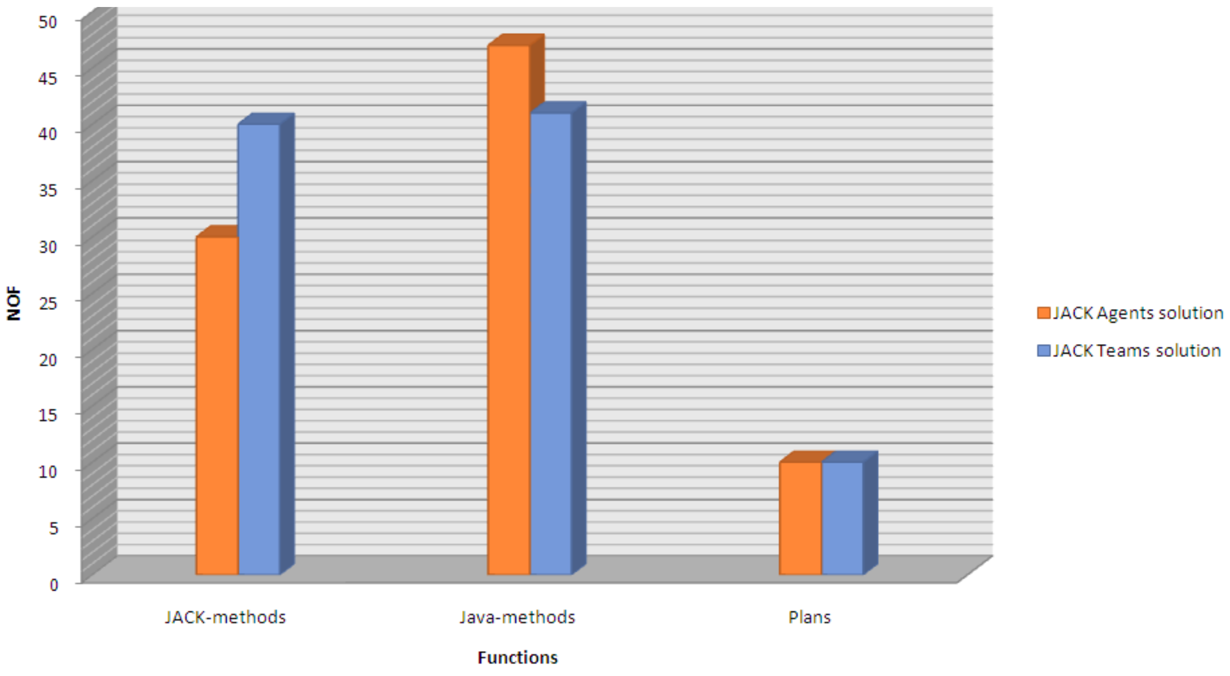


Figure 21: Measurements of Metric M3 (NOF)

Discussion

The diagram in Figure 21 shows that the subsea template package varies a lot between the two solutions. This is caused by JACK Agents solution having for-loops reusing code running through all teammembers. The JACK Teams solution reuses code by introducing a JACK-method to have repeated calls, working like a for-loop.

The JACK Agents version has 87 functions to realize the teamwork, while the JACK Teams version has 91 functions. This is an increase of 4.60%.

Conclusion

The result from the measurements of metric M3 shown in Figure 21 indicates that the number of functions will increase using JACK Teams compared to JACK Agents. Hypothesis H03 is therefore rejected and the alternative hypothesis HA3.2 is chosen.

9.1.4 Hypothesis 4

The hypothesis

Hypothesis 4 concerns the number of couplings between entities in each version and is given as follows:

- *H04:* Both versions will have the same number of couplings between the components in the system.
- *HA4.1:* The JACK Teams version will have fewer couplings between the components than JACK Agents version.
- *HA4.2:* The JACK Teams version will have more couplings between the components than JACK Agents version.

Measurements

Metric *M4 Number of Couplings between Entities (NOCBE)* is used in the testing of hypothesis 4. This metric represent the number of couplings between the agent/team and other entities. The packages looked into are the subsea template package and well package that realize teamwork differently in the two solutions. Coupling is defined in Subsection 4.1.5 to be in- and out going method calls and events. Method calls from an agent's own plan or posted events within the agent are not counted. Instanciating Java objects and JACK objects are not counted either. Events from the System events package sent from agents/teams/plans/teamplans are counted as they were included in these packages. The result of the measurements is found in Table 7.

M4: Number of Couplings between Entities (NOCBE)

External package	*Entities*	*JACK Agents version*				*JACK Teams version*			
		Coupling to subsea teamplate agent	Coupling from subsea template agent	Coupling to well agent	Coupling from well agent	Coupling to subsea template team	Coupling from subsea template team	Coupling to well team	Coupling from well team
Subsea template	Agent	-	-	0	0	-	-	0	0
	Plans	-	-	5	5	-	-	5	0
	Roles	-	-	0	0	-	-	0	0
	Java classes	-	-	0	0	-	-	0	0
Total Number of Couplings		**-**	**-**	***5***	***5***	**-**	**-**	***5***	***0***
Well	Agent	0	5	-	-	0	0	-	-
	Plans	5	0	-	-	0	0	-	-
	Roles	0	0	-	-	0	15		
	Java classes	0	13	-	-	0	15	-	-
Total Number of Couplings		***5***	***18***	**-**	**-**	***0***	***30***	**-**	**-**
Scenario structures	Java classes	0	27	0	35	0	27	0	35
Total Number of Couplings		***0***	***27***	***0***	***35***	***0***	***27***	***0***	***35***
Main-method	Java classes	20	0	48	0	40	0	72	0
Total Number of Couplings		***20***	***0***	***48***	***0***	***40***	***0***	***72***	***0***
Java library	Java classes	0	74	0	136	0	92	0	139
Total Number of Couplings		***0***	***74***	***0***	***136***	***0***	***92***	***0***	***139***
All packages		***25***	***119***	***53***	***176***	***40***	***149***	***77***	***174***

Table 7: Results for M4: Number of Couplings between Entities (NOCBE)

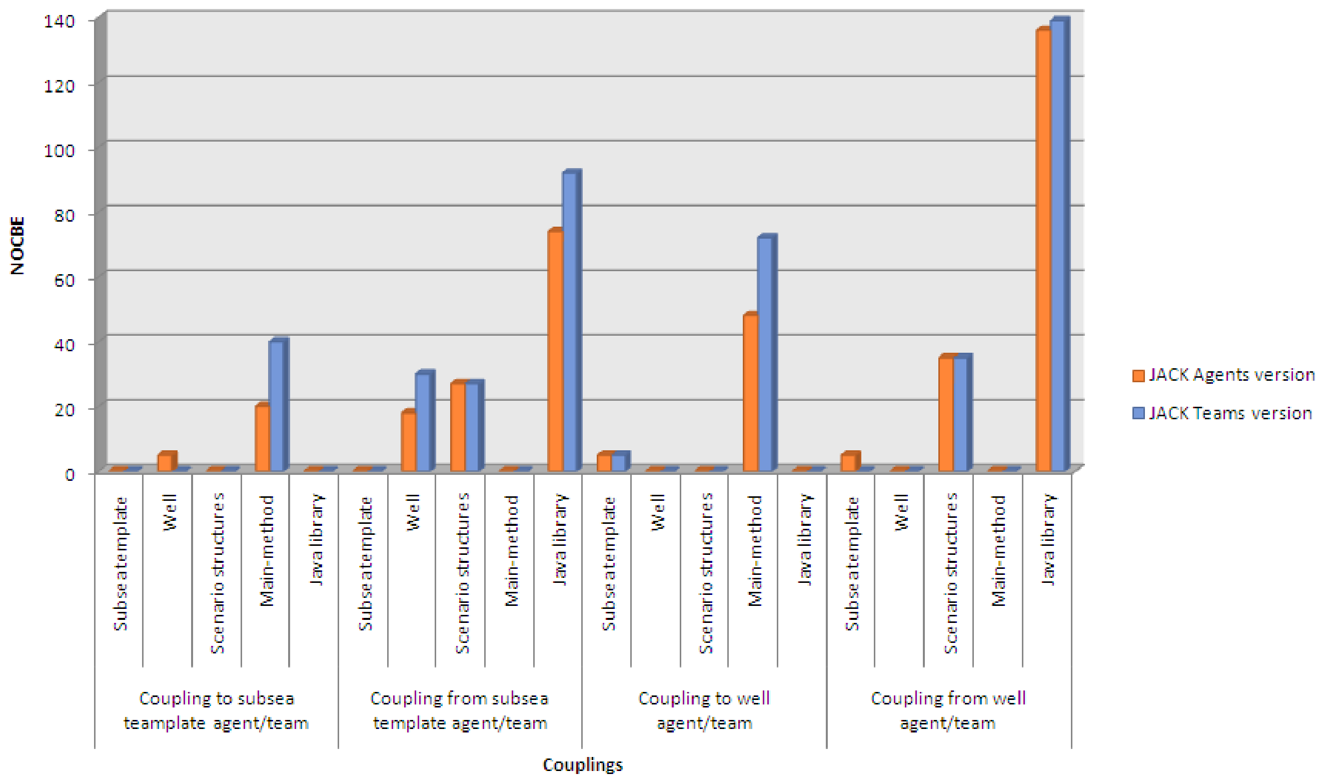


Figure 22: Measurements of Metric M4 (NOCBE)

Discussion

The diagram in Figure 22 shows that the different packages vary regarding couplings between the two solutions.

Coupling <u>to</u> the subsea template agent/team varies between the JACK Agents version and the JACK Teams version. The well package has no couplings because of the use of @teamAchieve. The JACK Teams version does not need to send an event to return data, because of the data exchange through the @teamAchieve statement. The result is no coupling. This is a decrease of 100%. The JACK Teams require more couplings to Java libraries mainly because of checking the status when instanciating teams. This is caused by a bug in the framework (see Section 9.3). This is an increase of 100%.

Coupling <u>from</u> subsea template agent/team varies between the JACK Agents version and the JACK Teams version. In the well package, this is mainly caused by using @teamAchieve statements to each role, instead of using a for-loop to run through all receivers. This is an increase of 66.67%. The Java library package has more couplings in the JACK Teams solution because of the establishment of teams and teammembers. This is an increase of 24.32%.

Coupling <u>to</u> well agent/team varies between the JACK Agents version and the JACK Teams version. The JACK Teams require more couplings to Java libraries mainly because of checking the status when instanciating teams, caused by a bug in the framework (see Section 9.3). This is an increase of 50%.

Coupling <u>from</u> well agent/team varies between the JACK Agents version and the JACK Teams version. The subsea template package has no coupling in the JACK Teams version, because it uses the @teamAchieve statement which creates no extra return-event like JACK Agents when doing data exchange. This is cause a decrease of 100% in number of couplings. The Java library package is slightly different in the two versions compared. The JACK Teams version causes an increase of 2.21%. The small difference in the Java library package is of no significance, and is probably caused by different coding style or inaccurate measurement.

Conclusion

The results from the measurements of metric M4 are shown in Figure 22. The important thing is to look at the JACK couplings and not at the Java couplings counted in the Java library package, Scenario structures package and the Main-method package. The data-exchange in JACK Teams creates fewer couplings, but the role-use creates more couplings. They approximately neutralize eachother in this experiment, and the null hypothesis H04 is therefore chosen. There is no proven or unproven proportional dependency between the two types of entitites in this experiment, and that makes it hard to generalize about.

9.1.5 Hypothesis 5

The hypothesis

Hypothesis 5 concerns the number of couplings between entities in each version and is given as follows:

- *H05:* The JACK Agents version and The JACK Teams version have the same number of external operations changing their internal state.

- *HA5.1:* The JACK Teams version has a fewer external operations changing the internal state than JACK Agents version.
- *HA5.2:* The JACK Teams version has larger amount of external operations changing the interna state than JACK Agents version.

Measurements

Metric *M5 Number of External Activations (NOEA)* is used in the testing of hypothesis 5. This metric represents the incoming couplings that activated the subsea template agent/team and the well agent/team. The results are shown in Table 8. The couplings were defined as external method-calls and received events in Subsection 4.1.5. The result is presented in a diagram in Figure 23 that illustrates the measurements related to each version.

M5: Number of External Activations (NOEA)

External package	*Entities*	*JACK Agents version*		*JACK Teams version*	
		External activations of subsea teamplate agent	External activations of well agent	External activations of subsea template team	External activations of well team
Subsea template	Agent	-	0	-	0
	Plans	-	5	-	5
	Roles	-	0	-	0
	Java classes	-	0	-	0
Total Number of Activations		**-**	***5***	**-**	***5***
Well	Agent	0	-	0	-
	Plans	5	-	0	-
	Roles	0	-	0	
	Java classes	0	-	0	-
Total Number of Activations		***5***	**-**	***0***	**-**
Scenario structures	Java classes	0	0	0	0
Total Number of Activations		***0***	***0***	***0***	***0***
Main-method	Java classes	20	48	40	72
Total Number of Activations		***20***	***48***	***40***	***72***
Java library	Java classes	0	0	0	0
Total Number of Activations		***0***	***0***	***0***	***0***
All packages		***25***	***53***	***40***	***77***

Table 8: Results for M5: Number of External Activations (NOEA)

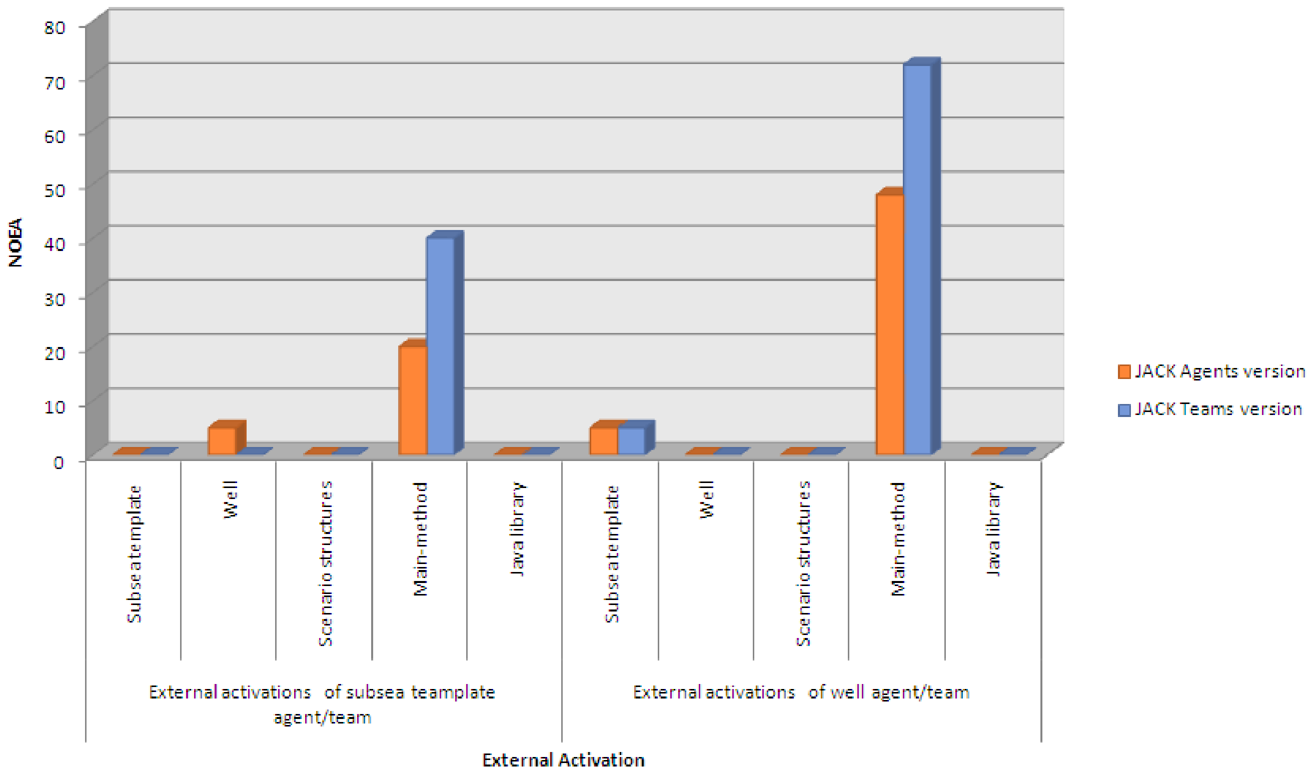


Figure 23: Measurements of Metric M5 (NOEA)

Discussion

The diagram in Figure 23 shows that the different packages vary regarding external activations.

External activations of the subsea template agent/team vary between the JACK Agents version and the JACK Teams version. The well package has no couplings because the use of the @teamAchieve statement. It causes no return-event to exchange data, and therefore results in no coupling. This is a decrease of 100% in the JACK Teams version. The JACK Teams require more couplings to Java libraries mainly because of checking the status when instanciating teams, caused by a bug in the framework (see Section 9.3). This is an increase of 100%.

External activations of the well agent/team vary between the JACK Agents version and the JACK Teams version. The JACK Teams version requires more couplings to Java libraries mainly because of checking the status when instanciating teams, caused by a bug in the framework (see Section 9.3). This is an increase of 50% in the JACK Teams version.

Conclusion

The result from the measurements of metric M5 is shown in Figure 23. The Main-method package in the JACK Teams version has an increased number of external operations, mainly caused by a bug in the framework (see Section 9.3). The number to put into focus is therefore the number of external operations between JACK-entities. The JACK Teams version will have fewer external operations changing the internal state regarding JACK-entities, compared to the JACK Agents version. Hypothesis H05 is therefore rejected and the alternative hypothesis HA5.1 is chosen.

9.1.6 Hypothesis 6

The hypothesis

Hypothesis 6 concerns about the level of abstraction in each version and is given as follows:

- *H06:* Use of JACK Teams will not provide a higher abstraction level for modeling and implementation of teamwork in an oil production system, compared to JACK Agents.
- *HA6.1:* Use of JACK Teams will provide a higher abstraction level for modeling and implementation of teamwork in an oil production system, compared to JACK Agents.

Measurements

A quantifying metric cannot be used to test hypothesis 6. Qualitative assessment is therefore used. The *modelings* of the two versions are considered according to the following definition for abstraction:

"…abstraction of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. In nearly all modern systems, elements interact with each other by means of interfaces that partition details about an element into public and private parts (35)*."*

When *implementing* the two versions, the term "abstraction level" substituted by the term "level of programming language[7] ". The "level of programming language" is defined as follows:

"In computing, a high-level programming language is a programming language with strong abstraction from the details of the computer. In comparison to low-level programming languages, it may use natural language elements, be easier to use, or more portable across platforms. Such languages hide the details of CPU operations such as memory access models and management of scope. A high level language isolates the execution semantics of a computer architecture from the specification of the program, making the process of developing a program simpler and more understandable with respect to a low-level language. The amount of abstraction provided defines how 'high level' a programming language is. (36)*"*

Discussion

JACK Teams has constructs that are made to support teamwork. It uses for instance belief propagation/inheritation and can use the @teamAchieve statement to exchange data between team-instances. Details are hidden, demanding no events doing the data exchange. The role-construct creates an extra encapsulation that can create reuse. The terminology used is more intuitive than JACK Agents when constructing teamwork. The possible composition of team formation is separated into a .def-file which hides the initial team structures from the rest of the code.

JACK is a cross-platform development environment written in Java. The programming language is far from CPU operations, which can be seen in the code structure. Source code must be contained in

[7] JACK Agents and JACK Teams are referred to as both modelling paradigms and programming languages in this report.

JACK entities. JACK Teams is an extension of JACK Agents, extending the support of contstructing teamwork. JACK Teams has therefore higher-level programming constructs than JACK Agents in construction of teamwork.

Conclusion

Based on the experiences made when developing the two versions, hypothesis HO6 is rejected and hypothesis HA6.1 is chosen.

9.2 Summary of results

Subsection 4.1.5 draws the connections between the hypotheses and benefits, which will be used to summarize the results of the hypotheses.

The development effort is concerned with the hypotheses 1-3. The JACK Agents version has 1175 lines of code to realize the teamwork, while the JACK Teams version has 1310 lines of code. This is an increase of 11.49%. The JACK Teams version will have more entities than the JACK Agents version. This is due to the increased number of role-entities when adding more teammember. The JACK Agents version has 87 functions to realize the teamwork, while the JACK Teams version has 91 functions. This is an increase of 4.60%. This increase of functions is due to extra functions needed to establish teams in JACK Teams. The experiment indicates that JACK Teams will cause a larger development effort than JACK Agents.

Reducing coupling is the concern in hypothesis 4. The data-exchange in JACK Teams creates fewer couplings, but the role-use creates more couplings. They approximately neutralize eachother in this experiment. This makes it hard to generalize about, but this experiment showed both versions having the same degree of couplings.

Encapsulation of functionality is concerned with the hypothesis 5. The number to put into focus is the number of external operations between JACK-entities. The JACK Teams version will have fewer external operations changing the internal state regarding JACK-entities, compared to the JACK Agents version. The well package has no couplings because of the use of the @teamAchieve statement. It does not have to send an event to return data, which leads to no coupling.

High abstraction level is the concern in hypothesis 6. JACK Teams has constructs made to support teamwork. It uses for instance belief propagation/inheritation and the @teamAchieve statement to exchange data between teams-instances. Details are hidden, needing no return-events doing data exchange. The role creates an extra encapsulation that can create reuse. The terminology used is more intuitive than JACK Agents when constructing teamwork. The possible composition team formation, is separated into an own .def-file which hides the possible team structures from the rest of the code. JACK Teams has therefore a higher-level programming constructs than JACK Agents in construction of teamwork.

Summarized, JACK Teams have the advantages of better encapsulation of functionality and higher abstraction level. If much data, more than the amount of required roles needed, is exchanged through the use of the @teamAchieve statement, the reduced coupling can be an advantage. If a large amount of roles are needed and less data exchange is done through @teamAchieve, this is a disadvantage using JACK Teams. The disadvantages are higher development effort, and possibly a higher amount of couplings.

9.3 A bug in the framework

A bug was discovered in JACK Teams during development of the JACK Teams solution. There was a problem assigning subteams to roles. AOS, the company developing the JACK framework, found a race condition in the framework causing the failure. A solution to work around the problem was to check that all the team-instances have instanciating before moving on in the main-method executing the program. The condition was checked by looking at the state of instanciation: team.getState() != Team.INITIAL_STATE.

9.4 Validity Concerns

Validity threats were described in Subsection 4.1.4. Some threats were accepted and other addressed to handle. The validity of the final results is as follows:

- ***Conclusion Validity***
  - *Low statistical power:* Only one version of each modeling paradigm was compared. Some of the conclusion may therefore be made with lack of data.
  - *Reliability of measures:* Metrics are used testing the hypotheses. They are quantitative and therefore objective, except hypothesis 6.
- ***Internal Validity***
  - *Selection:* The selection of objects may not be representative for all possible outcomes.
- ***Construction Validity***
  - *Experiment construction:* The experiment is constructed with define measurements, hypotheses and treatments. The relation between theory and observation is tried to be clearified.
  - *Mono-operation bias:* The quasi-exeriment constructed in this report may not show the whole picture of the theory.
- ***External Validity***
  - *Interaction of selection and treatment:* A student has constructed and performed the experiment, and may not be representative for the software developer population the results are generalized about.
  - *Interaction of setting and treatment:* Development tools and method used are up to date in order to make the experimental setting representative for the software industry.

Chapter 10

Qualitative results

This chapter describes differences in JACK Agents and JACK Teams in accordance to challenges set by the reference problem presented in Chapter 5. The qualitative approach compares the JACK Agents and JACK Teams solutions developed in the work of the experiment described in Section 4.1. Advantages and disadvantages between the two modelling paradigms are identified and explained. This will give more depth to the comparison of the two modelling paradigms, in addition to the quantitative approach. The evaluation considered the two aspects defined in Section 4.2, autonomy and scalability.

10.1 Autonomy

Autonomy is evaluated according to the construction of teamwork. The following subsections run through how the system design is related to autonomy, how it is constructed using JACK Agents and JACK Teams, and at the end a comparison of advantages/disadvantages between the two solutions implementing autonomy. Autonomy is defined in Subsection 4.2 to be *"the need for decisions to be made at any time, with some appreciation for the circumstance of the current situation (often referred to as situation awareness)."*

10.1.1 System design

The *delegation of autonomy machine to machine* is depending on the level of hierarchy one is looking at. The hierarchy presented in the oil production system is shown in the layered arhictecture presented in Figure 7. The field has the autonomy to choose between *proactive* and *reactive* system states, and initiate all sequences of actions accordingly. All teams below have a more limited autonomy, but are able to choose its production-contribution in *proactive state*. This selection and aggregation of production-scenarios reduce the volume of data on the production-scenarios presented for the human operator to pick among. A second effect is that production-scenarios are formatted into production-scenarios at different abstraction levels, for instance well production-scenario and subsea template production scenarios. The top-level team makes all decisions if the system state is in a *reactive state*. This is done in order to maintain a globally optimized solution.

The *delegation of human to machine autonomy* is more relevant in the planning phase than during the production phase, in the system design documented in this report. Input from the human operator is required in the planning phase, and the only feedback from the machine to the human operator during the production phase, is only the actual production hour by hour.

10.1.2 JACK Agents

The task that is being delegated to the teammembers cannot fail in the system designed in the work documented in this report. The @send statement is that is used to communicate between teams will always succeeds (because the event is sent asynchronously), and does not pick up failure of subteams. To wait for the event to be handled by the subteam, the @wait_for statement should be used in the team. The team will then wait for the subteam to return a message-event using the @reply statement. Delegation of autonomy is therefore depending on the trust that subteams does not fail. The @wait_for statement has to be used in order to check for some condition, if a team shall notice a failure among the teammembers.

An agent-address is needed in order to send events to it. The agent-address of the event-receiver can be found by registering the address inside the sender-agent at some point. Delegation of autonomy is used by sending events to the same agents every time, since the team structure is static due to the reference problem described. The psysical components in the reference problem will have the same dependencies to eachother, and therefore creating a static team structure.

The operator assistant uses the @waitFor() statement to get the human operator involved team processes. JACK Teams is not used towards the human operator in the solutions developt. Only JACK Agents are, or teams-instances used exactly like an agent-instance. A team-instance is an extension of an agent-instance because the JACK Teams modelling paradigm is an extension of the JACK Agents modelling paradigm. JACK Teams was decided to focus on machine to machine autonomy delegation between the subsea template and well level, since it was perceived to be a better way to show the potential of JACK Teams in this experiment. The interface towards human operators was considered during development to be implemented in the same way in both JACK Agents and JACK Teams.

10.1.3 JACK Teams

A team-instance has subteams-instances to fulfil the different roles, and is responsible to coordinate the action of the teammembers. The team-instance uses a teamplan to get subteams-instances to fulfil its teamgoals and subtasks. The @teamAchieve statement is used to send a message event to the subteams-instances, through the roles. When the subteam-instance that fulfil a role fails to do the task, it can propagate (if not handled in the subteam) the exception back to the team-instance by using the @teamAchieve statement. This was not used in the system designed in this report, because all agents/teams are programmed not to fail by a fixed sequence of serial actions. The @teamAchieve(roleinstance_ref.peer, EventInstance) statement is used to send event back from the subteam-instance to the team-instance.

Delegation of work is done through roles, which the subteams-instances fulfil. The team formations need to be static in order to access the data regarding a specific physical component, because the different subteams-instances contain the needed datastructures to store data about the specific physical component. Each role can just be fulfilled by one subteam-instance at the time. Each subteam-instance therefore has to fulfil seperate roles in the teams-instances, which created extra role-entities needed. Creating extra teams-instances during runtime and add them as new teammembers would be a problem since role-entities are defined at compilation time. Adding new teams-instances fulfilling roles in the .def-file (contains all possible team formations) was experienced to be possible. Adding new teams (not removing) was tested in a small example besides the experiment. A team-instance should also be able to change what roles it can perform during runtime. However, the author of this report did not examine this during the use of the JACK framework.

One thing that was observered that could benefit dynamic teams was belief propagation and inheritation between teams-instances and teammembers-instances. The software developer does not have to care with implementing all the event-traffic data exchange causes. The data exchange is coded more or less straight into the beliefset. This could be a suited way to copy data back and forth to give the subteam-instances all the data needed to fulfil the role, if different teams-instances fulfil the same role at different times during runtime. This type of data exchange solves the problem of distributed data needed by others. This was tried implemented in the solutions documented in this report, but it failed to run properly. The cause of the failure was not identified, and should be further investigated. Example-code that came along with the JACK framework product showed how belief propagation/inheritation could be implemented in smaller example, and indicated some of the benefits described in this paragraph.

10.1.4 Comparison

The comparison with advantages and disadvantages given in this subsection are based on the system design (see Chapter 6) documented in this report, and not necessarily valid in the general case. Table 9 shows the advantages and disadvantes between the two solutions developed.

	JACK Agents version	*JACK Teams version*
Advantages	Supports a static team structure with having address-lists in the team, containing the addressed of the connected teammembers.	Built-in support to construct a teamstructure of roles and to declare what subteam-instance that can fulfil what roles needed by a specific team-instance.
		Built-in propagation of exception from teammembers.
		Using a JACK Teams .def-file type to set up possible teamstructures, instead of Java-methods gives a better overview of the team-hierarchy.
		Propagation and inheritance of beliefs do not create any extra events to be sent.
		Built-in support for dynamically allocate teams to roles. Copy data needed by the new teammember by using belief propagation and inheritance.
		Roles work as interfaces and make it easier to see what is needed of events to be sent and to be handled.
		The @teamAchieve statement waits until it is either successful or not (asynchronously). Does not need to wait for another event in return.
Disadvantages	Asynchronously has to wait for (using the @wait_for statement) a returned event after using the @send statement. The return-event is sent by using the @reply statement. The @teamAchieve statement in JACK Teams gives better team-support because it is synchronously.	Using a simple texteditor to generate the .def-file is a bit difficult in the work of getting all the special symbols typed correctly. JACOB[8] could have been used, but was not in this work.
	Implementing human operator interaction requires what was considered to be relatively much coding, and could possibly been made simpler in an improved framework.	Implementing human operator interaction probably has to done usin the same approach as in the JACK Agents modeling paradigm.

Table 9: Autonomy comparison

10.2 Scalability

Scalability is evaluated according to scalability related to construction of teamwork. The following subsections run through how the system design is related to scalability, how it is realized using JACK Agents and JACK Teams, and at the end a comparison of advantages/disadvantages between the two solutions. Scalability is defined in Subsection 4.2 to be the development effort when expanding the system with more instances of the different agent- or teams constructs.

[8] The JACOB™ Object Modeller (JACOB) is a system providing machine and language independent object structures (42).

10.2.1 System design

The reference problem defined in this report contains the number of physical compontents the system design should handle. The structure is hierarchical, and if new physical components are added will they be placed in one of the hierarchical levels that already exist. The performance is not supposed to decrease much because the distributed work enables parallel reasoning. The team-commander has to do more work if it has more teammembers connected to it. A quantitative evaluation on how an increased number of teammembers affects the performance is not performed in this work. This is however an important issue in order to get global optimization. A large amount of computations that generates the production-scenarios are needed, and an increased number of teammembers will affect the performance of the system.

10.2.2 JACK Agents

Adding agent-instances to the different hierarhical levels requires registration of the new agent-address in its belonging team-commander agent. Scaling the system therefores create little extra work and can occur during compilation and runtime.

10.2.3 JACK Teams

A few things need to be done in order to add new teams into the different hierarchical levels of the existing team structure. An increased number of role-entities are needed when more teammembers are introduced. This is a result of each role only being fulfilled by one team-instance at the time, and teammembers working in parallel have to fulfil their own separate role. It is however possible to perform reasoning based on the actual team membership if needed, because the team can access its possible sub-teams through the role container. This will cause a type of interaction more like the one JACK Agents uses.

Adding new teams-instances fulfilling roles has to be updated in the .def-file (contains all possible team formations). This was experienced to be possible. Adding new teams (not removing) was tested in a small example besides the experiment. A team-instance should be able to change what roles it can perform during runtime, but the author did not experience if this could be done in a satisfying manner.

10.2.4 Comparison

The comparison with advantages and disadvantages given in this subsection are based on the system design (see Chapter 6) constructed in this report, and not necessarily valid in the general case. Table 10 shows the advantages and disadvantes between the two solutions developed.

	JACK Agents version	***JACK Teams version***
Advantages	Only register a new teammember in the team-commander.	A team-instance can perform several roles at the same time.
Disadvantages	Difficult to keep track on what teammember is allowed to do what kind of work.	Needs extra role entities when new teammembers working in parallel are introduced. It is however possible to perform reasoning based on the actual team membership if needed.
		Each role can only be fulfilled by one team at a time. Since roles are defined at compile time, this limits the number of teammembers operating at the same time.
		Updating what team that is allowed to fulfill which roles, can be a possibly limitations of the JACK Team modeling paradigm. This was not examined in the work documented in this report and should be further investigated.

Table 10: Scalability comparison

Chapter 11

Summary of work

This project was motivated by the applicability and suitability using JACK Agents or JACK Teams to construct teamwork. It was expected that JACK Teams would be more feasible than JACK Agents, because it is a modelling paradigm specially designed to support the costruction of teamwork. JACK Teams is an extension of JACK Agents, and can be used in the same way as JACK Agents if wanted. The elements extended to JACK Teams can be listed as follows:

- Teamdata
- Belief propagation and inheritance
- Exception propagation
- Dynamic team formations
- Use of @teamAchieve

Teamdata enables beliefs to be propagation and inheritation between teams and subteams. Subteams can propagate belief to the the team, and subteam can inheritate beliefs from a team. Teamdata is contained in the team and shall reflect the common beliefs that will be consistent with the subteams' belief. Propagation and inheritation of beliefs in JACK Agents, would propably create extra event-traffic if beliefs were distributed in the system.

Belief propagation was tried implemented, but did not work in the JACK Teams solution constructed. The cause of failure was not discovered. An example that came along with the JACK framework product showed that belief propagation and belief inheritance in JACK Teams run asynchronously. Asynchronously means that beliefs are propagated or inheritated while the user-plan is still running. This creates parallelism that can possibly be exploited in for instance distributed systems that runs large amouts of computation. The belief propagation and inheritation also enables beliefs to be copied back and forth between teams and subteams. Copy of beliefs is needed when different teams fulfil a role during runtime. The teams will then have access to the needed data to fulfil the role, by using belief propagation and inheritation.

Exception propagation propagates Java exceptions to the team if not handled in the subteam. Note that this is not the same as a plan failing. This feature was not examined.

Team formation consists of two things: the possible teamstructures defined at compile time, and task teams established within these teamstructures to solve a tasks during runtime. The possible teamstructures has to be defined in a .def-file at compile time (it can possibly be updated during runtime, but this was not tested). Task teams are dynamic teams that have the ability to change what subteam that fulfils what role, during runtime. The relationships between the physical components in the reference problem are static, and did not use this feature to the fully extend.

A *role* only can be fulfilled by one team at the time. Subteams that want to operate at the same time within the team need to fulfil different roles to be able to work simultaneously. In JACK Teams, this created several role-entities. It is however possible to perform reasoning based on the actual team membership if needed, because the team can access its possible sub-teams through the role container.

The *@teamAchieve statement* activates subteams by sending an event to the subteam. The team that sent event by using the @teamAchieve statement then waits until the event has been processed by the subteam. Changes in the event made by the subteam can be maintained by the team when the @teamAchieve statement is finished. This feature therefore enables data exchange, whitout any returning event that contains the data.

Chapter 12

Conclusion

The problem definition described in Chapter 1 presented three questions:

1. Is it easier to develop teamwork in JACK Teams than in JACK Agents, when building a decision-support system?
2. Is it possible to develop a similar system in JACK Agents and JACK Teams?
3. Will JACK Teams be a more feasible platform than JACK Agent, when developing teamwork?

The *first question* is looking at the aspects of development effort. The experience is that JACK Teams require a bit extra development effort compared to JACK Agents. This extra effort is caused by extra lines of code. An increase of 11.49% of code lines in the JACK Teams version was measured. The increase of functions was measured to be 4.60%. The extra number of functions was a result of the additional functions needed to establish teams in JACK Teams.

The *second question* looks at the output for the two versions developed. Both versions had the same static composition of teams and teammembers. The algorithms performing all the computations that resulted in the output-values, shown in the graphical user interface, were the same in both versions. The teamstructure and distribution of work was the same in both versions. The most important difference between the two versions was how the communication was performed between team-instances and agent-instances. The JACK Teams version required less event-trafic doing data exchange, but needed extra JACK entities because of the introduction of the role-entities. It is however possible to perform reasoning based on the actual team membership if needed, because the team can access its possible sub-teams through the role container. This was not implemented in the JACK Teams version, and therefore causing a different result. Both version did however implement the same type of system, and performed the same results.

The *third question* asked if JACK Teams is a more feasible platform for development than JACK Agents, when implementing teamwork. The data-exchange in JACK Teams creates fewer couplings, but the role-use creates more couplings. They approximately neutralize eachother in this experiment. Introducing more teammembers, could have caused a large number of role-entities needed and therefore extra couplings. The external operations between JACK entities decreased, and resulted in better encapsulation of functionality. JACK Teams has constructs specially design to support teamwork construction. It uses for instance belief propagation/inheritation and uses @teamAchieve to exchange data between teams. Details are hidden, demanding no events for the data exchange. The role creates an extra encapsulation that can enable reuse. The terminology used is more intuitive than JACK Agents when constructing teamwork. The possible composition of team formations is separated to be contained in a special .def-file. This declaration hides the possible formations from the rest of the code. JACK Teams is therefore considered by the author to provide a higher-level programming constructs than JACK Agents. Introducing teams in large scale was not performed, and could have caused a problem if each team is supposed to fulfil its own separate role defined in a role-entity. Considering all the above-mentioned elements, and the increase development effort needed in the use of JACK Teams, caused JACK Agents to be considerate as the most feasible platform for development of teamwork in this case.

The main conclusion is that systems with static team formation will cause JACK Agents to be the preferred modeling paradigm. Dynamic team formations during runtime were not needed due to the reference problem introduced. Maintaince during runtime, for instance introducing new subteams and changing the role structure was not looked into. Introducing teams in large scale was not performed. These four factors could have caused a different result. The question is if JACK Teams shows its potential through the oil production system designed in this report.

Chapter 13

Future work

This report has looked at teamwork created by JACK Agents and JACK Teams, using the reference problem described in Chapter 5 as application area. Future work on the *oil production system design* is to implement parallel and continous working process-loops, not serial and discrete as the present one (one by one hour). The system should be able to have parts of the system running in proactive state and having other parts running in a reactive state at the same time. Different levels of predictability should also be taken into account. The global optimization technique with all the computation on the different abstraction levels needed should be validated with a more realistic number of physical components and real life data model used. Prediction of production used in the proactive state should be continuously updated, to be as precisely as possible. The possible advantage created by the use of proactive planning compared to reactive adjustements should be investigated.

Human-centric systems possess the possibility for the agent system to learn from humans, and therefore enable agents to become so-called *human-centric smart agents*. These systems can have interaction in both directions. A future system should look at how this learning process can be implemented to take a new step towards human-centric systems.

What concerns JACK Teams is how parallelism can be increased by using the *belief propagation and inheritation* supported. The belief propagation and inheritation also enables beliefs to be copied back and forth between teams and subteams. Copy of beliefs is needed when different teams fulfil a role during runtime. The teams will then have access to the needed data to fulfil the role, by using belief propagation and inheritation. This should be examined and confirmed.

The *roles a team can fulfil* were static and remained the same during runtime. How to make change this role-structure during runtime should be investigated. A well can for example change from being a production well to being an injection well during runtime. The well should therefore loose its possibility of fulfing a production role, and be able to fulfil an injection role.

The *human operator can operate with different levels of autonomy*. Further work should look into how the human operator can operate towards teams contained in different hierarchical levels in the system. An important question is how different roles affect eachother in combination with different degrees of autonomy varying during runtime. Several human operators should be able to use the system at the same time. How they work together with different levels of autonomy must be looked into in order to realize this simulatenous work between human operators. The agent system shall also vary the degree of autonomy itself, and human operator should be able to manually adjust the delegation of autonomy. The escalation and de-escalation of the level of autonomy is import in the work to involve the human operator a satisfying manner in the right situations on the right point in time.

Dynamic teams should be looked into, using human operators (with different delegation of autonomy), and also other elements like of economy and oil transportation can be taken into consideration. This may show the full potential of JACK Teams.

Part IV

Appendices

Appendix A

“Steps of action”-scenarios

This appendix describes pseudo algorithms of the “steps of action”-scenarios presented in Subsection 6.3.2.

A.1 “Steps of action”-scenario one - Planned and predicted production

Objective: Plan a configuration that meets the production target within a fixed period of time.

Situation: Production is being planned and put into action.

Algorithm:

1. *Operator Assistant* sends a number of hours the production shall be reached within.
2. *Field Planning & Monitoring* asks *Geographical Area Planning & Monitoring* to generate possible production-scenarios of geographical areas, within the period of time chosen by *Operator Assistant.*
3. *Geographical Area Planning & Monitoring* asks *Subsea Template Planning & Monitoring* to generate possible production-scenarios of subsea templates, within the period of time chosen by *Operator Assistant.*
4. *Subsea Template Planning & Monitoring* asks *Well Planning & Monitoring* to generate possible production-scenarios of wells, within the period of time chosen by *Operator Assistant.*
5. *Well Planning & Monitoring* sends predicted well production-scenarios to *Subsea Template Planning & Monitoring.*
6. *Subsea Template Planning & Monitoring* combines all well production-scenarios, and chooses the three (number of production-scenarios is set to three to simplify and decrease the number of combinations) best production-scenarios. These are sent to *Geographical Area Planning & Monitoring.*
7. *Geographical Area Planning & Monitoring* combines all subsea template production-scenarios, and chooses the nine (amount of production-scenarios is set to nine to simplify and decrease the number of combinations) best production-scenarios. These are sent to *Field Planning & Monitoring.*
8. *Field Planning & Monitoring* generates field production-scenarios, and sends all production-scenarios to *Operator Assistant.*
9. *Operator Assistant* chooses a field production-scenario. The production-scenario chosen is told *Field Planning & Monitoring.*

10. *Field Planning & Monitoring* tells *Geographical Area Planning & Monitoring* which production-scenario that is chosen.

11. *Geographical Area Planning & Monitoring* tells *Subsea Template Planning & Monitoring* which production-scenario that is chosen.

12. *Subsea Template Planning & Monitoring* tells *Well Planning & Monitoring* which production-scenario that is chosen.

13. *Well Planning & Monitoring* implements the production-scenario chosen, start the production according to the well settings required by the production-scenario chosen.

14. *Field Planning & Monitoring* asks *Geographical Area Planning & Monitoring* to monitor the production after one timestep.

15. *Geographical Area Planning & Monitoring* asks *Subsea Template Planning & Monitoring* to monitor the production after one timestep.

16. *Subsea Template Planning & Monitoring* asks *Well Planning & Monitoring* to monitor the production after one timestep.

17. *Well Planning & Monitoring* checks if production is according to the well production-set. If yes, step 14-17 are repeated until until production target is reached, or the period of time selected is finished. If no, the system switchs to production-scenario number two: "Unpredicted changes according to planned production".

A.2 "Steps of action"-scenario two - Unpredicted changes according to planned production

Objective: Meet production target within the period of time set.

Situation: Production is not going according to the predicted and planned production, because of reservoir dynamics.

Algorithm:

1. *Well Planning & Monitoring* tells *Subsea Template Planning & Monitoring* it did not follow the production-scenario it said it would produce according to.

2. *Subsea Template Planning & Monitoring* tells *Geographical Area Planning & Monitoring* it did not follow the production-scenario it said it would produce according to.

3. *Geographical Area Planning & Monitoring* tells *Field Planning & Monitoring* it did not follow the production-scenario it said it would produce according to.

4. *Field Planning & Monitoring* tells *Geographical Area Planning & Monitoring* to switch to "reactive adjustement" process.

5. *Geographical Area Planning & Monitoring* tells *Subsea Template Planning & Monitoring* to switch to "reactive adjustement" process.

6. *Subsea Template Planning & Monitoring* sends *Geographical Area Planning & Monitoring* the oil/waste ratio for the last hour for the specific production level.
7. *Geographical Area Planning & Monitoring* sends *Field Planning & Monitoring* the subsea template oil/waste ratio for the last hour for the specific production level.
8. *Field Planning & Monitoring* chooses the subsea templates with best oil/waste ratios until the processing facility capacity is fulfilled. *Field Planning & Monitoring* sends the list of chosen production levels for the different subsea templates to *Geographical Area Planning & Monitoring.*
9. *Graphical Area Planning & Monitoring* sends the list of chosen production levels for the different subsea templates to *Subsea Template Planning & Monitoring.*
10. *Subsea Template Planning & Monitoring* sends *Well Planning & Monitoring* the different production levels to implement.
11. *Field Planning & Monitoring* asks *Geographical Area Planning & Monitoring* to monitor the production after one timestep.
12. *Geographical Area Planning & Monitoring* asks *Subsea Template Planning & Monitoring* to monitor the production after one timestep.
13. *Subsea Template Planning & Monitoring* asks *Well Planning & Monitoring* to monitor the production after one timestep.
14. *Well Planning & Monitoring* checks if the production target is reached, or the period of time selected is finished. The algorithm is then repeated if the target is not reached and period of time is not finished.

Appendix B

Design

This appendix shows the detailed design of both the JACK Agents- and JACK Teams solutions developed during the work documented in this report. The notation of the diagrams shown can be found in Appendix C.

B.1 JACK Agents solution

This appendix contains detailed design diagrams used in the development of the JACK Agents solution described in Chapter 7. Two "steps of action"-scenarios that were implemented in the solution were defined in Subsection 6.3.2: "*Planned and predicted production"* and "*Unpredicted changes according to planned production".*

The first "steps of action"-scenario "*Planned and predicted production"* is divided into several interaction sequences to easier see what happens during the scenario. The following subsections are divided into *"Generate production-scenarios"*, *"Choose production-scenario"*, and *"Monitor production against production-scenario"*.

The second "steps of action"-scenario "*Planned and predicted production"* is divided into several interaction sequences to easier see what happens during the scenario. The following subsections are divided into *Start reactive "well choke settings" state* and *Monitor reactive" well choke settings" state*.

B.1.1 Generate production-scenarios

The sequence diagram with description can be found in Subsection 7.2. Figure 24 shows how the JACK entities are connected to eachother.

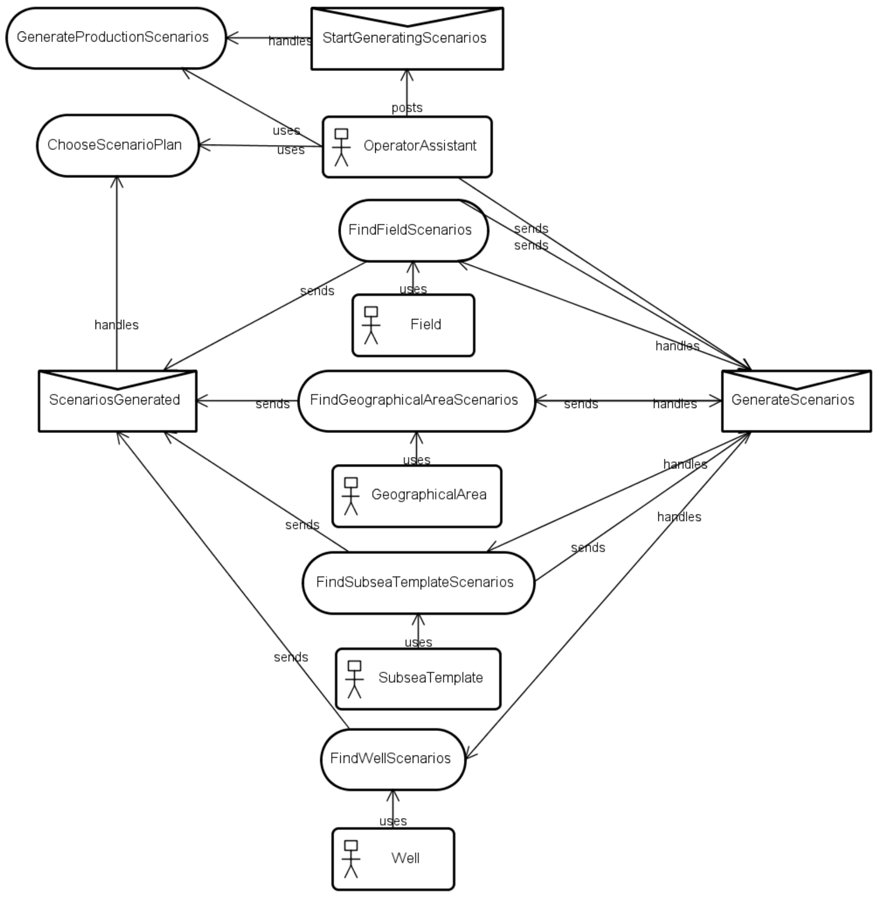


Figure 24: JACK entities involved in the "Generate production-scenarios"-scenario

B.1.2 Choose production-scenario

The scenario begins when the proactive planning process (see scenario: *generate production-scenarios*) has created production-scenarios with different production targets predicted. Figure 25 shows all agents, plans, and events involved in this process. Figure 26 shows how the JACK entities are connected to eachother.

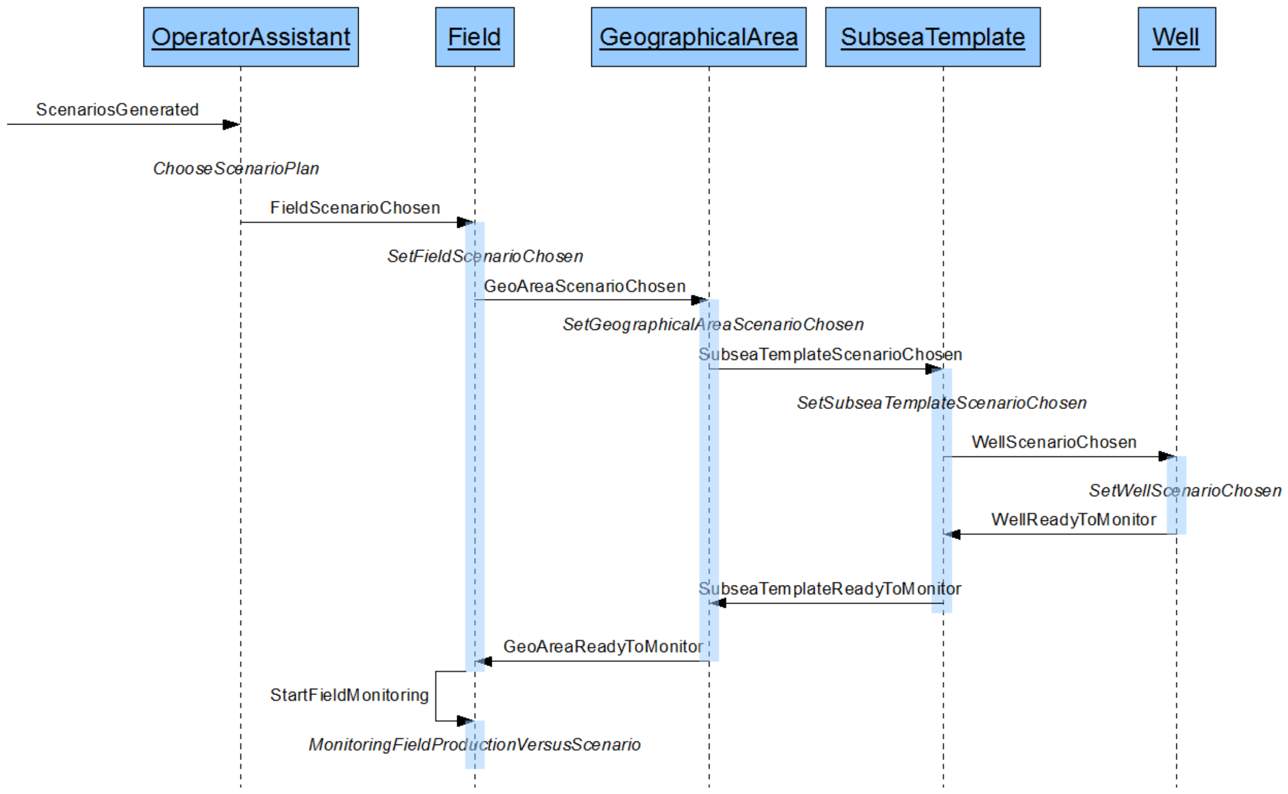


Figure 25: Choose production-scenario

The human operator has to specify a field production-scenario id to be choosed. This production-scenario is a composition of production-scenarios from the level belove in the system hierarchy. In this case the geographical area. The id's of the composite production-scenarios are found, and the decomposing production-scenario process is repeated until reaching the bottom-level well. The different well settings are the put into action, and production is started. The well propagates a message saying that monitoring can now start because the actual production has started.

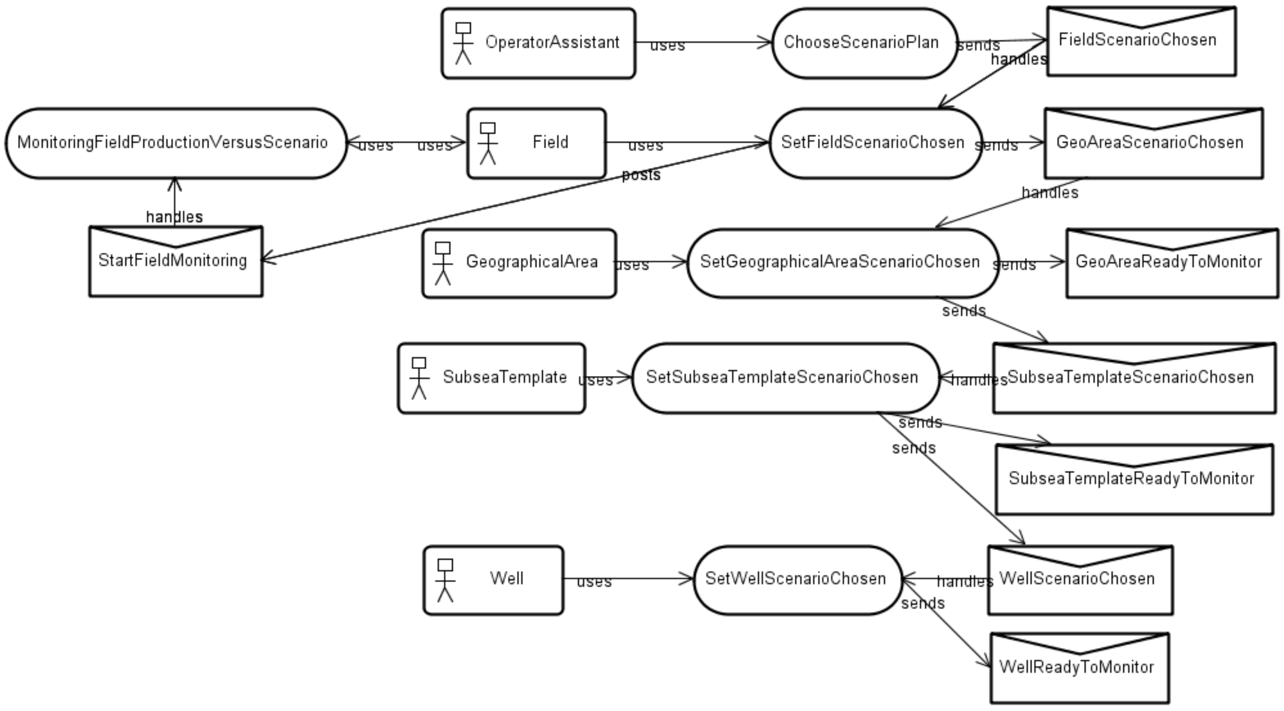


Figure 26: JACK entities involved in the "Choose production-scenario"-scenario

B.1.3 Monitor production against production-scenario

Monitoring production against production-scenario begins after the selected field production-scenario has been chosen (see scenario: *choose production-scenarios*) and the system is set accordingly. Figure 27 shows all agents, plans, and events involved in this process. Figure 28 shows how the JACK entities are connected to eachother.

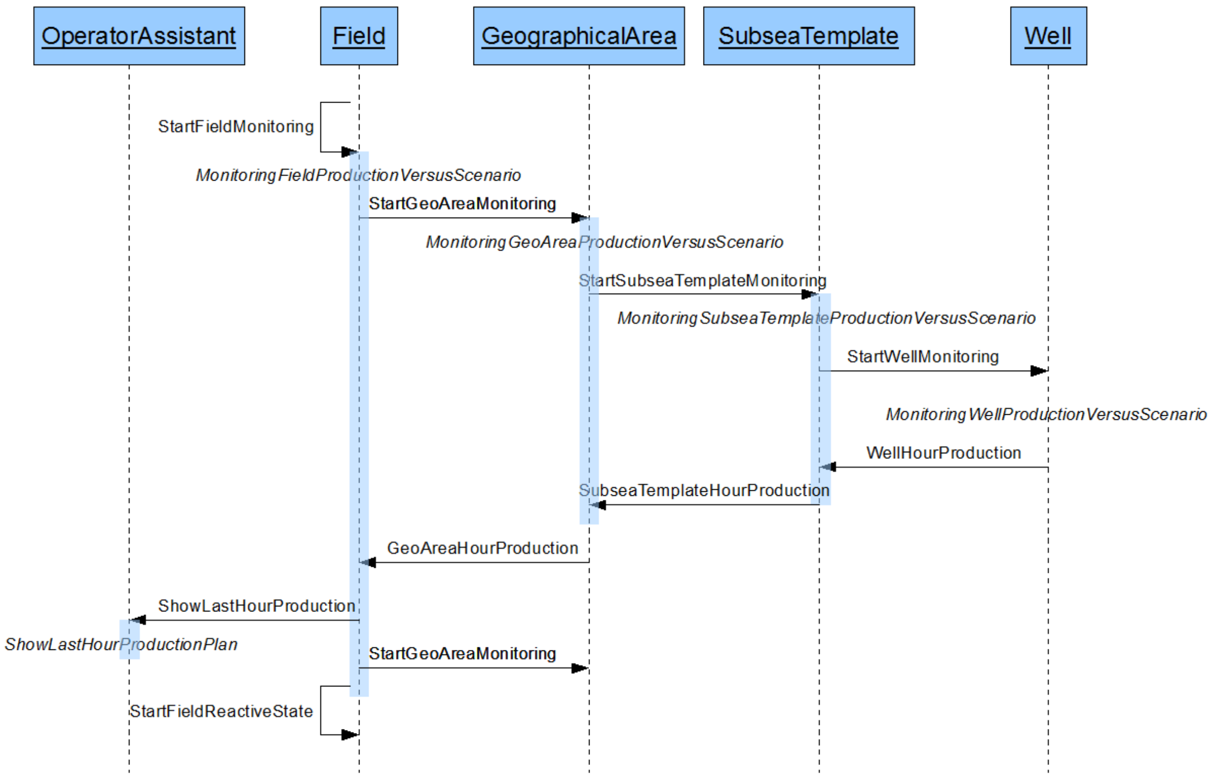

OperatorAssistant
Field
GeographicalArea
SubseaTemplate
Well
StartFieldMonitoring
MonitoringFieldProductionVersusScenario
StartGeoAreaMonitoring
MonitoringGeoAreaProductionVersusScenario
StartSubseaTemplateMonitoring
MonitoringSubseaTemplateProductionVersusScenario
StartWellMonitoring
MonitoringWellProductionVersusScenario
WellHourProduction
SubseaTemplateHourProduction
GeoAreaHourProduction
ShowLastHourProduction
ShowLastHourProductionPlan
StartGeoAreaMonitoring
StartFieldReactiveState

Figure 27: Monitor production against production-scenario

When monitoring is ready to start since the actual production has started, a sample to check actual production against production-scenarios planned are initiated by the field. The wanted timeinterval to check for accumulated production is sent downwards the hierarchy to the hierarchical bottom-level well. The amount of production and system state for the next hour is then propagated and combined in each level reaching the field level in the end. If one or more wells want to change from proactive to reactive, the whole system has to change to reactive. This is done by posting a StartFieldReactive in the field. If the system remains in a proactive state, the process is repeated for the next hour by sending a StartGeoAreaMonitoring-event, and keeps repeating if state remains proactive until the period of time is finished.

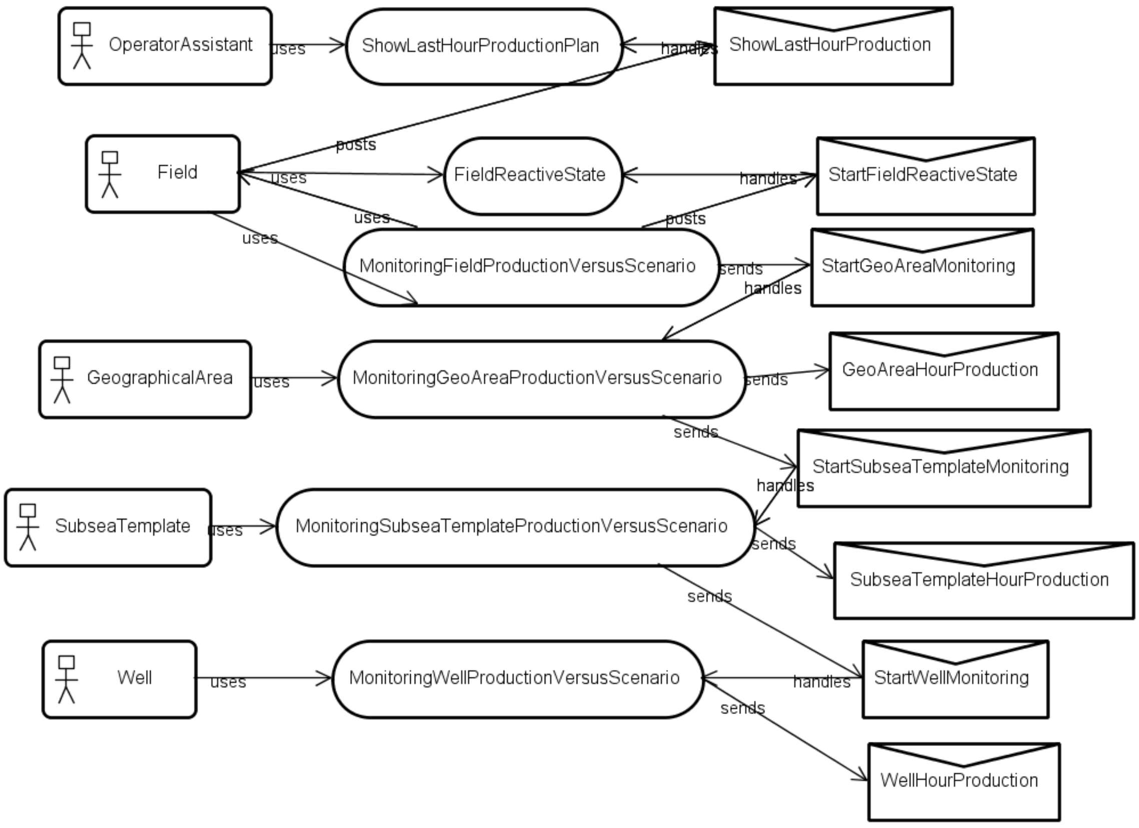


Figure 28: JACK entities involved in the "Monitor production against production-scenario"-scenario

B.1.4 Start reactive ” well choke settings” state

To change system state into reactive begins if the actual production and production-scenarios selected did not match (see scenario: *Monitor production against production-scenario*). Figure 29 shows all agents, plans, and events involved in this process. Figure 30 shows how the JACK entities are connected to eachother.

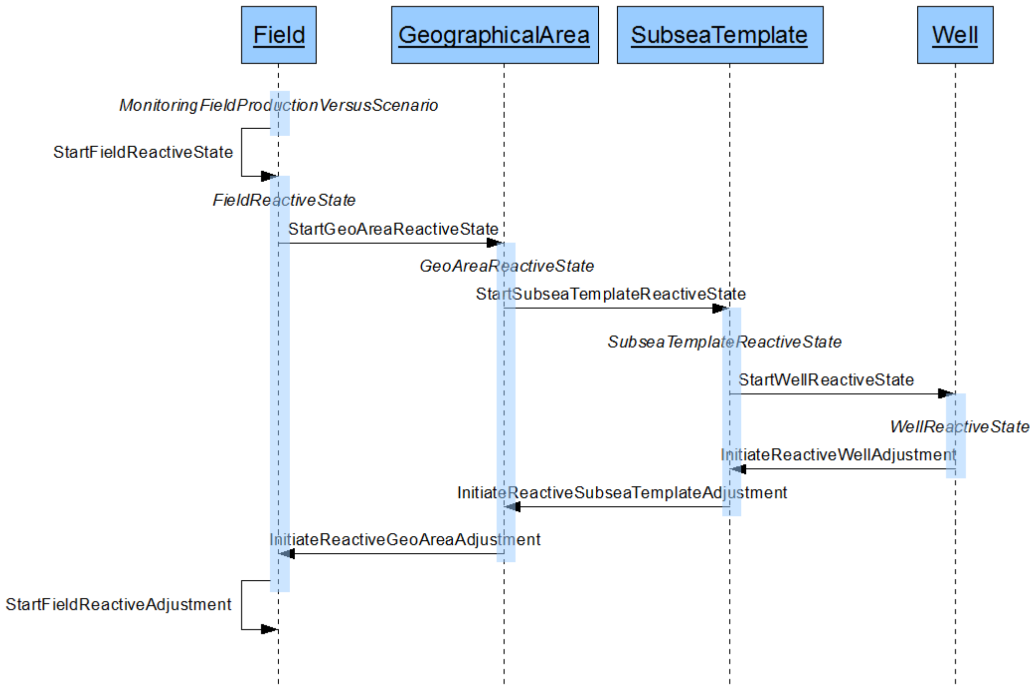

Field
GeographicalArea
SubseaTemplate
Well
MonitoringFieldProductionVersusScenario
StartFieldReactiveState
FieldReactiveState
StartGeoAreaReactiveState
GeoAreaReactiveState
StartSubseaTemplateReactiveState
SubseaTemplateReactiveState
StartWellReactiveState
WellReactiveState
InitiateReactiveWellAdjustment
InitiateReactiveSubseaTemplateAdjustment
InitiateReactiveGeoAreaAdjustment
StartFieldReactiveAdjustment

Figure 29: Start reactive ”well choke settings” state

If the system state has changed from proactive to reactive, the field will ask downwards the system to the production for the last hour. The last hour production is propagated to the subsea template team, where it is accumulated. This accumulation will show which well level (all well must is assumed to have the same well choke setting towards the subsea template) creates which oil/waste-ratio. Propagating the answer to the field, the next scenario *Monitor reactive" well choke settings" state* will have a look at how to choose what subsea templates will instructs its teammember with what well choke settings.

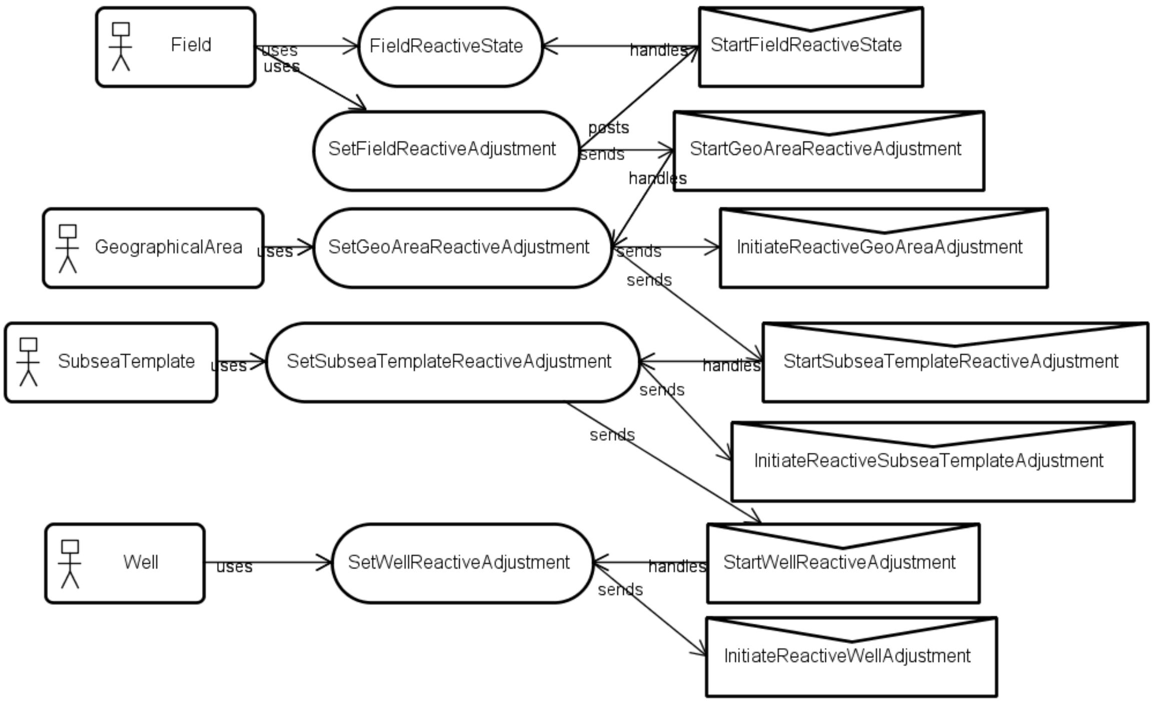

Field
uses
uses
FieldReactiveState
handles
StartFieldReactiveState
SetFieldReactiveAdjustment
posts
sends
StartGeoAreaReactiveAdjustment
handles
GeographicalArea
uses
SetGeoAreaReactiveAdjustment
sends
InitiateReactiveGeoAreaAdjustment
sends
SubseaTemplate
uses
SetSubseaTemplateReactiveAdjustment
handles
StartSubseaTemplateReactiveAdjustment
sends
sends
InitiateReactiveSubseaTemplateAdjustment
Well
uses
SetWellReactiveAdjustment
handles
StartWellReactiveAdjustment
sends
InitiateReactiveWellAdjustment

Figure 30: JACK entities involved in the "Start reactive *well choke settings* state"-scenario

B.1.5 Monitor reactive" well choke settings" state

After the reactive "well choke settings" state has started (see scenario: *Start reactive "well choke settings" state*), the newly calculated subsea template settings from field has be adjusted and have the system produce according to the new settings. Figure 31 shows all agents, plans, and events involved in this process. Figure 32 shows how the JACK entities are connected to eachother.

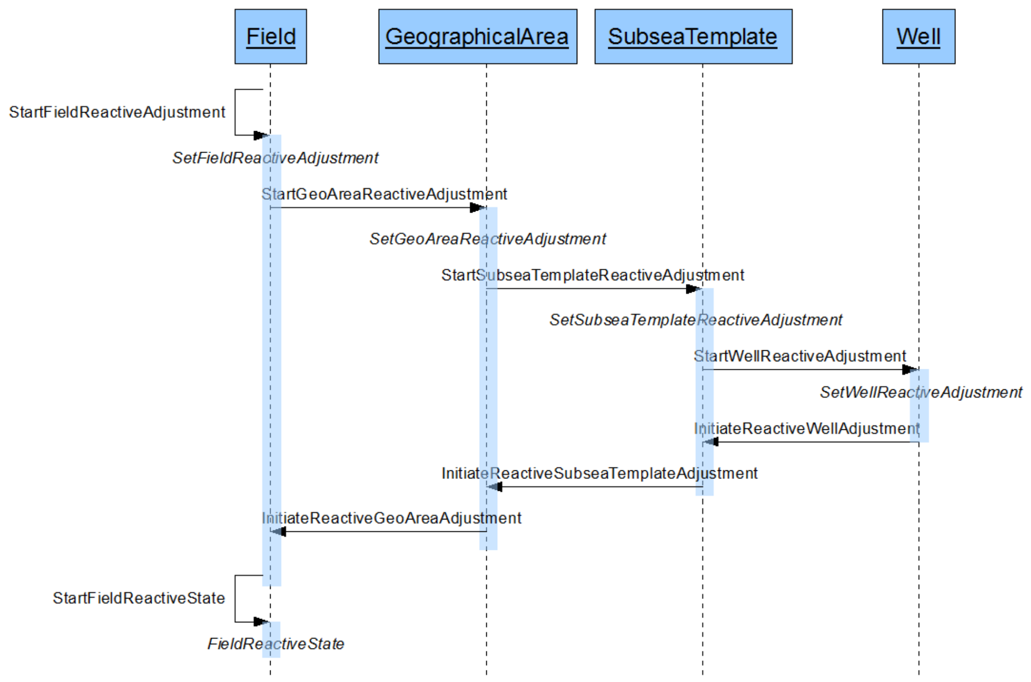

Field
GeographicalArea
SubseaTemplate
Well
StartFieldReactiveAdjustment
SetFieldReactiveAdjustment
StartGeoAreaReactiveAdjustment
SetGeoAreaReactiveAdjustment
StartSubseaTemplateReactiveAdjustment
SetSubseaTemplateReactiveAdjustment
StartWellReactiveAdjustment
SetWellReactiveAdjustment
InitiateReactiveWellAdjustment
InitiateReactiveSubseaTemplateAdjustment
InitiateReactiveGeoAreaAdjustment
StartFieldReactiveState
FieldReactiveState

Figure 31: Monitor reactive ”well choke settings” state

The StartFieldReactiveAdjustment-event contains oil/waste ratios for the subsea templates last hour’s production. In addition well choke levels +/- 1 is predicted with oil/waste-ratio and amount of production. The field then fills up the total capacity with the subsea templates having the best oil/waste-ratio, to maximize oil production every hour. Maximization every hour will be the best way to ensure that the production target is reached within the period of time. The field then propagates downwards in the system the new settings. The actual production is then monitored in the wells using the new well choke settings. The amount of production is then propagated to the field, which make a check if the total production target set by the human operator is reached. If not, the reactive state process is repeated again starting with the scenario: *Start reactive ”well choke settings” state.*

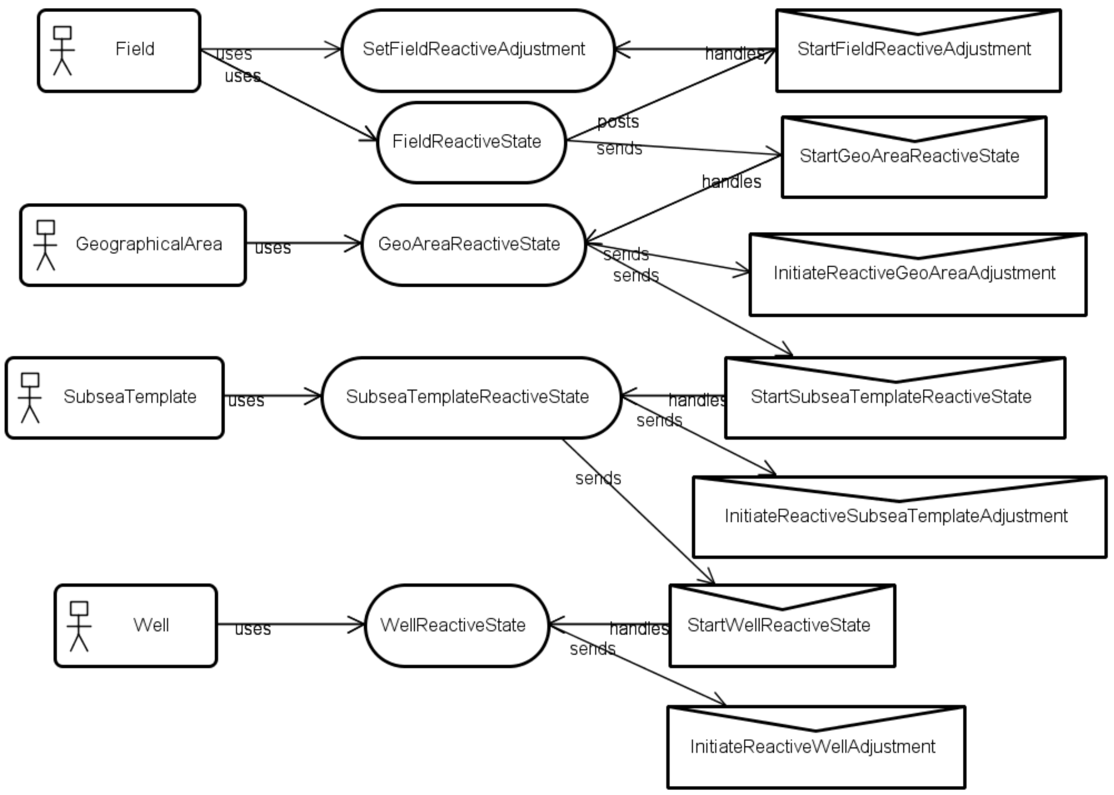

Field
uses
uses
SetFieldReactiveAdjustment
handles
StartFieldReactiveAdjustment
FieldReactiveState
posts
sends
StartGeoAreaReactiveState
handles
GeographicalArea
uses
GeoAreaReactiveState
sends
sends
InitiateReactiveGeoAreaAdjustment
SubseaTemplate
uses
SubseaTemplateReactiveState
handles
sends
StartSubseaTemplateReactiveState
sends
InitiateReactiveSubseaTemplateAdjustment
Well
uses
WellReactiveState
handles
sends
StartWellReactiveState
InitiateReactiveWellAdjustment

Figure 32: JACK entities involved in the "Monitor reactive *well choke settings* state"-scenario

B.2 JACK Teams solution

This appendix contains detailed design diagrams used in the development of the JACK Teams solution described in Chapter 8. Two “steps of action”-scenarios that were implemented in the solution were defined in Subsection 6.3.2: “*Planned and predicted production”* and “*Unpredicted changes according to planned production”.*

The first “steps of action”-scenario “*Planned and predicted production”* is divided into several interaction sequences to easier see what happens during the scenario. The following subsections are divided into *“Generate production-scenarios”*, *“Choose production-scenario”*, and *“Monitor production against production-scenario”*.

The second “steps of action”-scenario “*Planned and predicted production”* is divided into several interaction sequences to easier see what happens during the scenario. The following subsections are divided into *Start reactive ” well choke settings” state* and *Monitor reactive” well choke settings” state*.

B.2.1 Generate production-scenarios

The sequence diagram with description can be found in Subsection 8.2. Figure 33 shows how the JACK entities are connected to eachother.

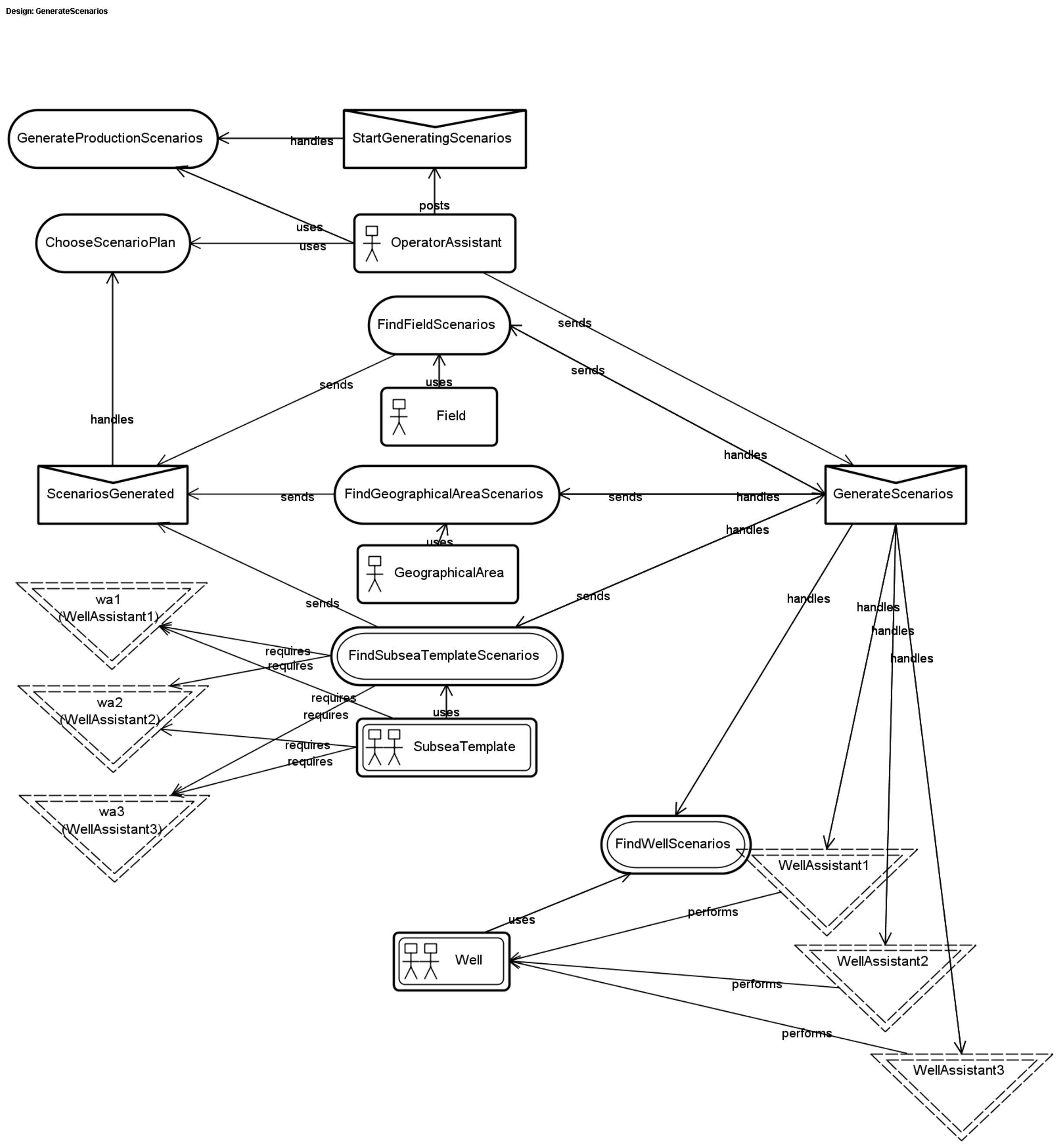


Figure 33: JACK entities involved in the "Generate production-scenarios"-scenario

B.2.2 Choose production-scenario

The scenario begins when the proactive planning process (see scenario: *generate production-scenarios*) has created production-scenarios with different production targets predicted. Figure 34 shows all agents, plans, and events involved in this process. Figure 35 shows how the JACK entities are connected to eachother.

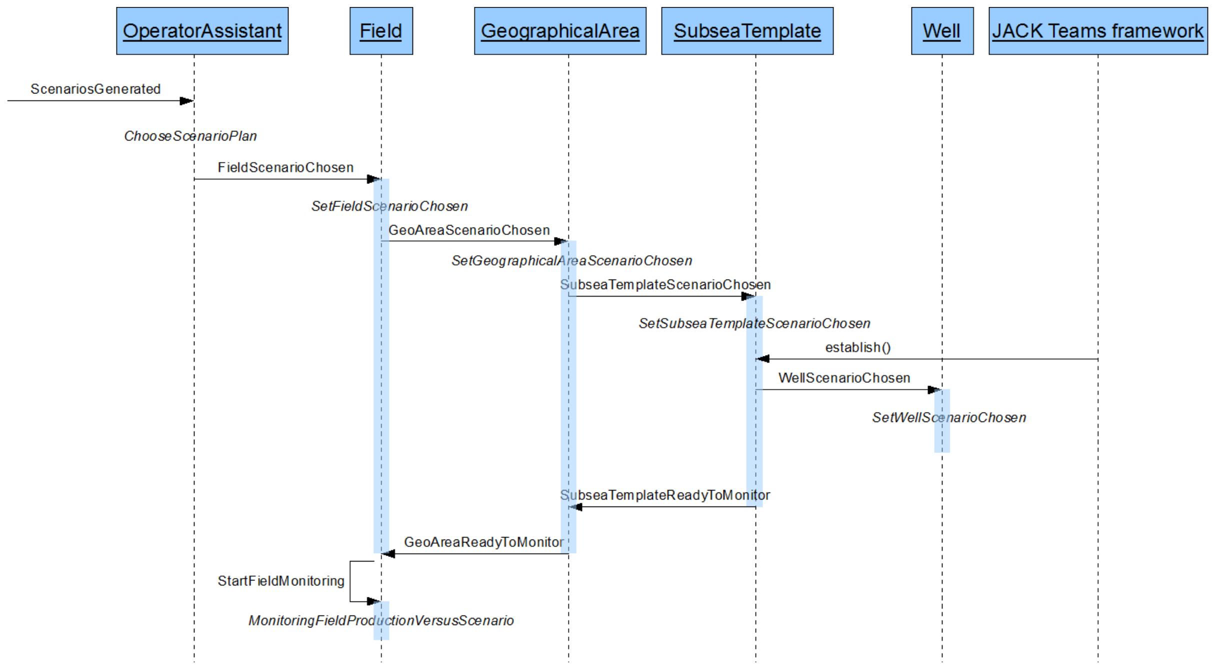


Figure 34: Choose production-scenario

The scenario starts with the human operator specifying a field production-scenario id to be choosed. This production-scenario is a composition of production-scenarios from the level belove in the system hierarchy. In this case the geographical area. The id's of the composite production-scenarios are found, and the decomposing production-scenario process is repeated until reaching the bottom-level well. The different well settings are the put into action, and production is started. The well propagates a message saying that monitoring can now start because the actual production has started.

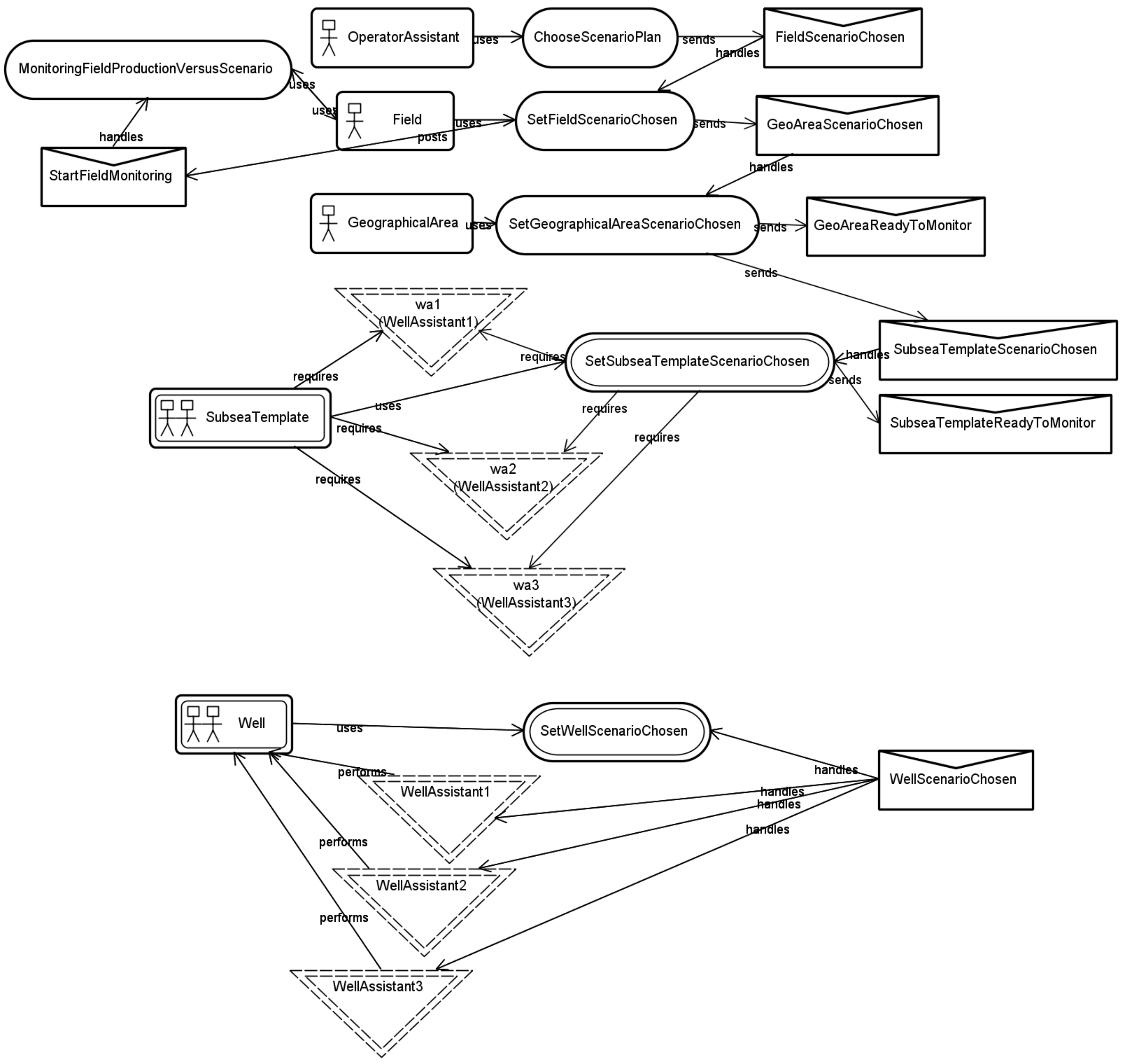


Figure 35: JACK entities involved in the "Choose production-scenario"-scenario

B.2.3 Monitor production against production-scenario

Monitoring production against production-scenario begins after the selected field production-scenario has been chosen (see scenario: *choose production-scenarios*) and the system is set accordingly. Figure 36 shows all agents, plans, and events involved in this process. Figure 37 shows how the JACK entities are connected to eachother.

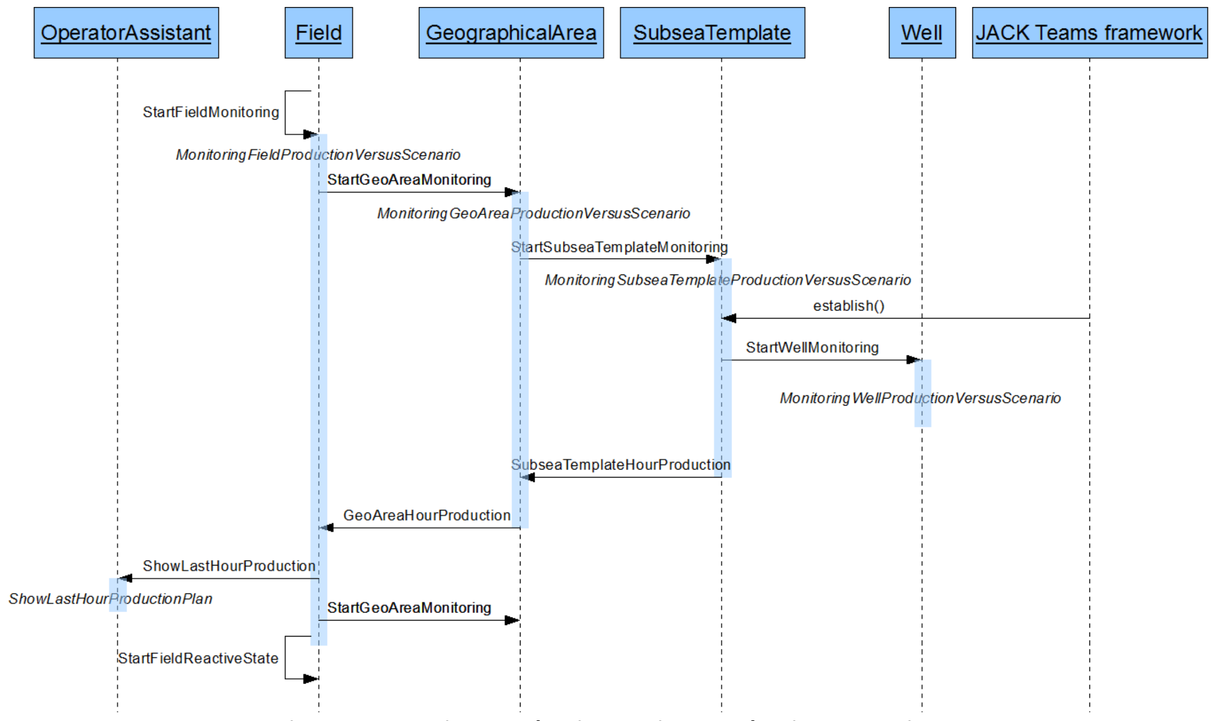

OperatorAssistant
Field
GeographicalArea
SubseaTemplate
Well
JACK Teams framework
StartFieldMonitoring
MonitoringFieldProductionVersusScenario
StartGeoAreaMonitoring
MonitoringGeoAreaProductionVersusScenario
StartSubseaTemplateMonitoring
MonitoringSubseaTemplateProductionVersusScenario
establish()
StartWellMonitoring
MonitoringWellProductionVersusScenario
SubseaTemplateHourProduction
GeoAreaHourProduction
ShowLastHourProduction
ShowLastHourProductionPlan
StartGeoAreaMonitoring
StartFieldReactiveState

Figure 36: Monitor production against production-scenario

When monitoring is ready to start since the actual production has started, a sample to check actual production against production-scenarios planned are initiated by the field. The wanted timeinterval to check for accumulated production is sent downwards the hierarchy to the hierarchical bottom-level well. The amount of production and system state for the next hour is then propagated and combined in each level reaching the field level in the end. If one or more wells want to change from proactive to reactive, the whole system has to change to reactive. This is done by posting a StartFieldReactive in the field. If the system remains in a proactive state, the process is repeated for the next hour by sending a StartGeoAreaMonitoring-event, and keeps repeating if state remains proactive until the period of time is finished.

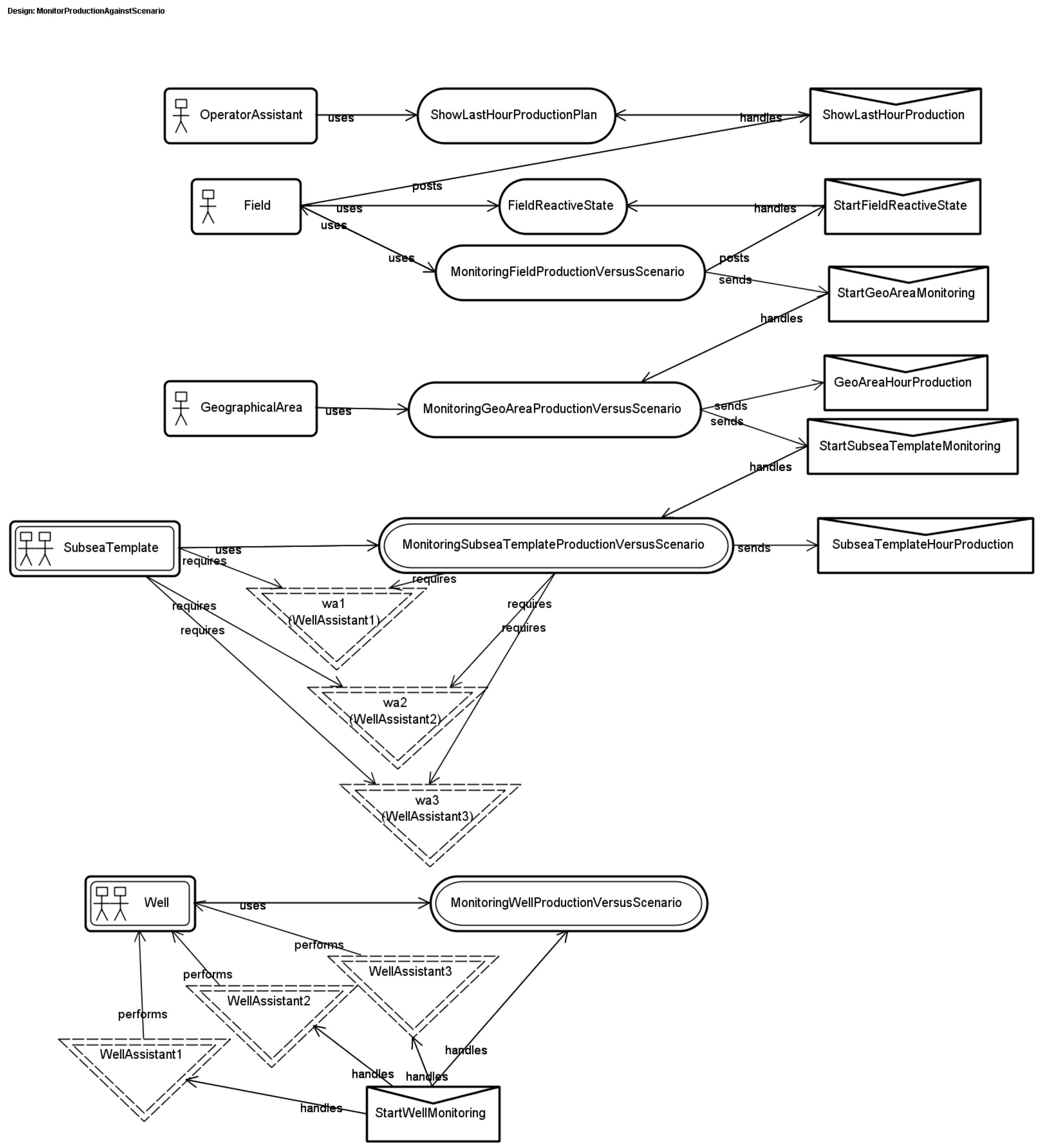


Figure 37: JACK entities involved in the "Monitor production against production-scenario"-scenario

B.2.4 Start reactive ” well choke settings” state

To change system state into reactive begins if the actual production and production-scenarios selected did not match (see scenario: *Monitor production against production-scenario*). Figure 38 shows all agents, plans, and events involved in this process. Figure 39 shows how the JACK entities are connected to eachother.

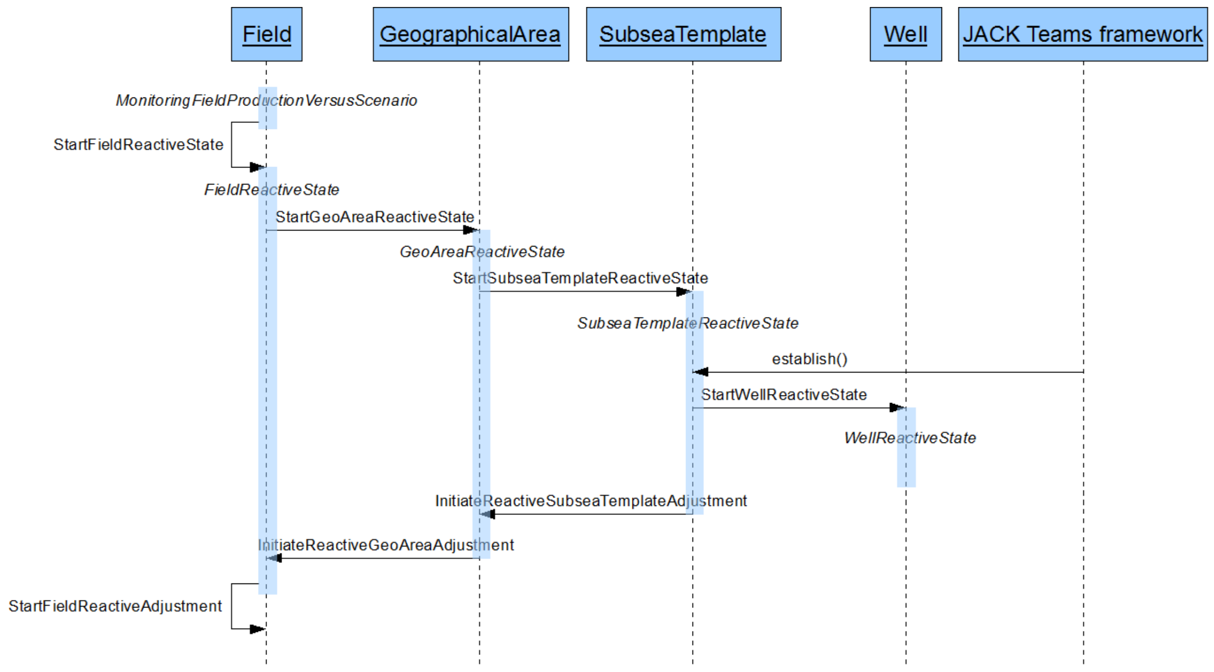

Field
GeographicalArea
SubseaTemplate
Well
JACK Teams framework
MonitoringFieldProductionVersusScenario
StartFieldReactiveState
FieldReactiveState
StartGeoAreaReactiveState
GeoAreaReactiveState
StartSubseaTemplateReactiveState
SubseaTemplateReactiveState
establish()
StartWellReactiveState
WellReactiveState
InitiateReactiveSubseaTemplateAdjustment
InitiateReactiveGeoAreaAdjustment
StartFieldReactiveAdjustment

Figure 38: Start reactive ”well choke settings” state

If the system state has changed from proactive to reactive, the field will ask downwards the system to the production for the last hour. The last hour production is propagated to the subsea template team, where it is accumulated. This accumulation will show which well level (all well must is assumed to have the same well choke setting towards the subsea template) creates which oil/waste-ratio. Propagating the answer to the field, the next scenario *Monitor reactive” well choke settings” state* will have a look at how to choose what subsea templates will instructs its teammember with what well choke settings.

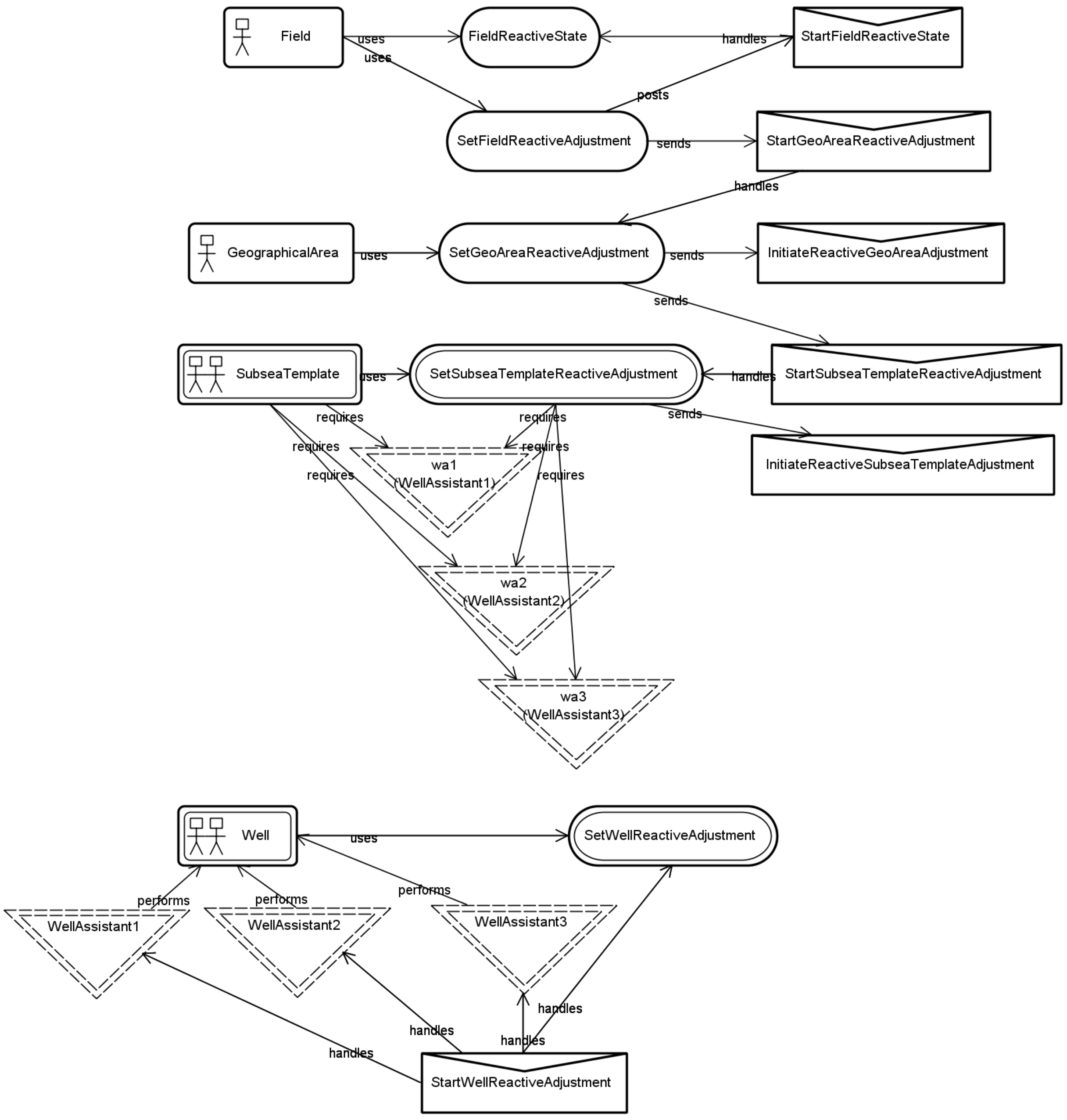

Field
uses
uses
FieldReactiveState
handles
StartFieldReactiveState
posts
SetFieldReactiveAdjustment
sends
StartGeoAreaReactiveAdjustment
handles
GeographicalArea
uses
SetGeoAreaReactiveAdjustment
sends
InitiateReactiveGeoAreaAdjustment
sends
SubseaTemplate
uses
SetSubseaTemplateReactiveAdjustment
handles
StartSubseaTemplateReactiveAdjustment
sends
InitiateReactiveSubseaTemplateAdjustment
requires
wa1 (WellAssistant1)
wa2 (WellAssistant2)
wa3 (WellAssistant3)
Well
uses
SetWellReactiveAdjustment
performs
WellAssistant1
WellAssistant2
WellAssistant3
handles
StartWellReactiveAdjustment

Figure 39: JACK entities involved in the "Start reactive *well choke settings* state"-scenario

B.2.5 Monitor reactive" well choke settings" state

After the reactive "well choke settings" state has started (see scenario: *Start reactive "well choke settings" state*), the newly calculated subsea template settings from field has be adjusted and have the system produce according to the new settings. Figure 40 shows all agents, plans, and events involved in this process. Figure 41 shows how the JACK entities are connected to eachother.

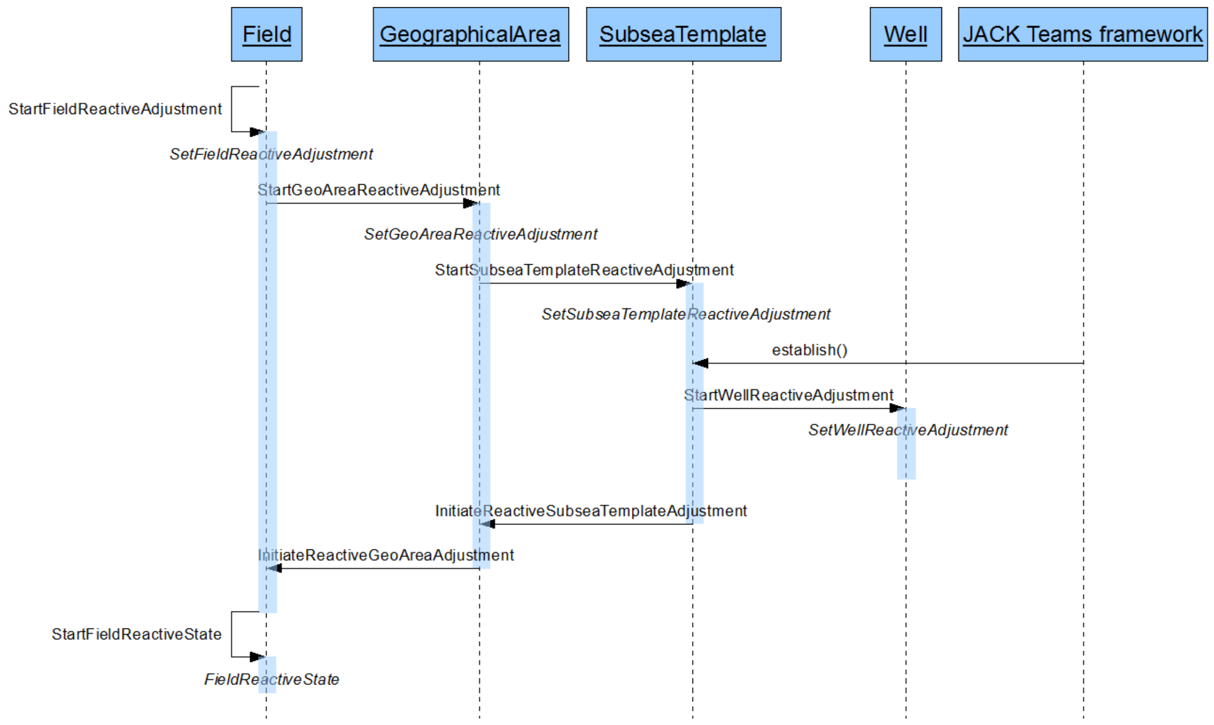

Field
GeographicalArea
SubseaTemplate
Well
JACK Teams framework
StartFieldReactiveAdjustment
SetFieldReactiveAdjustment
StartGeoAreaReactiveAdjustment
SetGeoAreaReactiveAdjustment
StartSubseaTemplateReactiveAdjustment
SetSubseaTemplateReactiveAdjustment
establish()
StartWellReactiveAdjustment
SetWellReactiveAdjustment
InitiateReactiveSubseaTemplateAdjustment
InitiateReactiveGeoAreaAdjustment
StartFieldReactiveState
FieldReactiveState

Figure 40: Monitor reactive" well choke settings" state

The StartFieldReactiveAdjustment-event contains oil/waste ratios for the subsea templates last hour's production. In addition well choke levels +/- 1 is predicted with oil/waste-ratio and amount of production. The field then fills up the total capacity with the subsea templates having the best oil/waste-ratio, to maximize oil production every hour. Maximization every hour will be the best way to ensure that the production target is reached within the period of time. The field then propagates downwards in the system the new settings. The actual production is then monitored in the wells using the new well choke settings. The amount of production is then propagated to the field, which make a check if the total production target set by the human operator is reached. If not, the reactive state process is repeated again starting with the scenario: *Start reactive "well choke settings" state.*

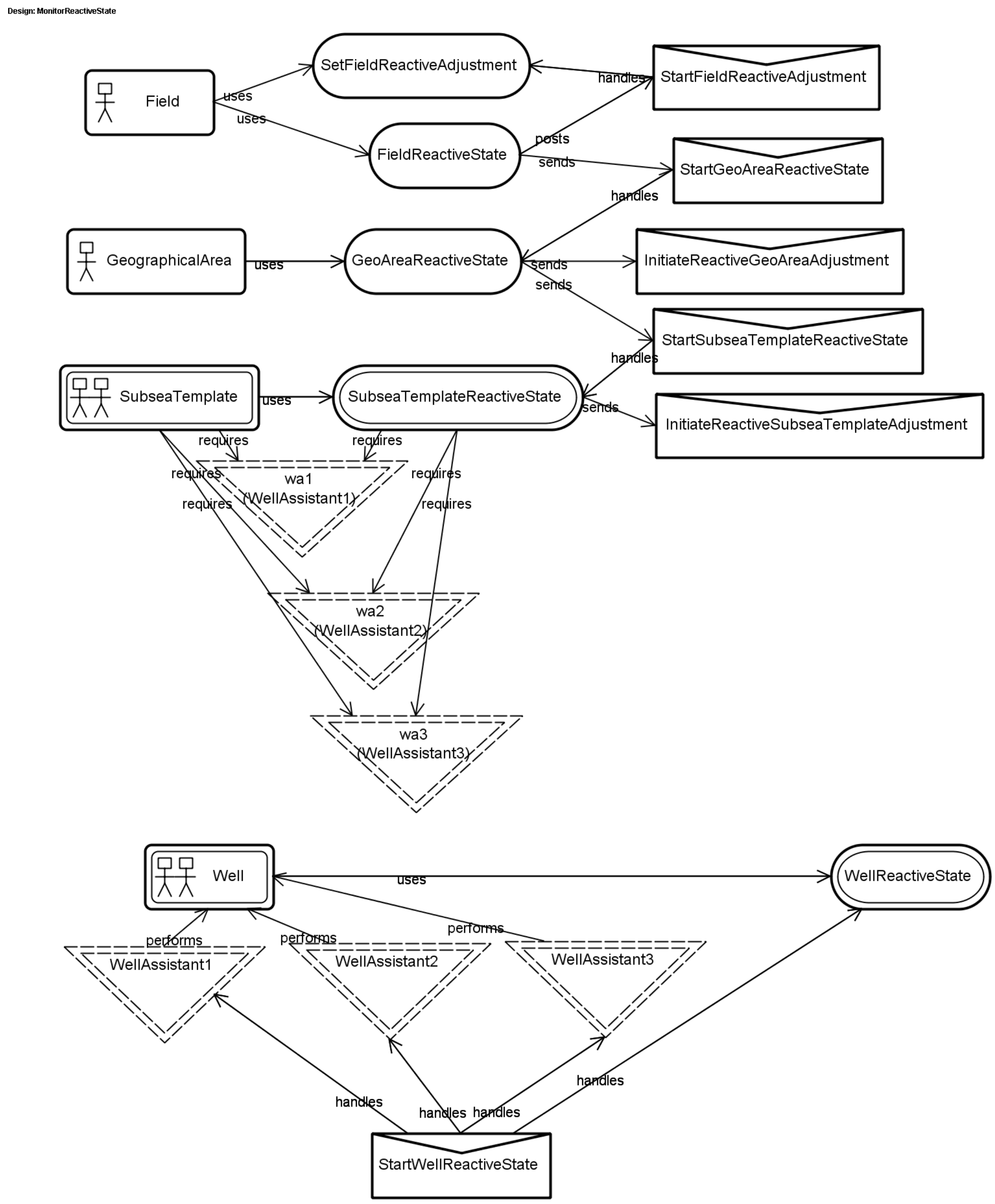


Figure 41: Monitor reactive *well choke settings* state

Appendix C

Notation

This appendix defines the notation used in the sequence diagrams and and JACK Development Environment (JDE) design diagrams shown in the report.

C.1 Sequence diagram

Table 11 describes the notation used in the sequence diagrams shown in the report.

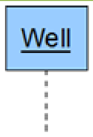
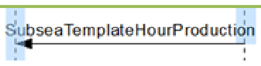
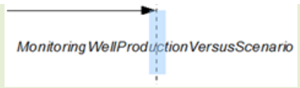
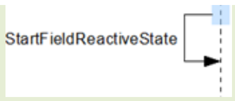
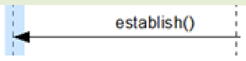
	The blue box represent agent or team		The arrow with text represent an event sent from one agent/team to another agent/team
	The blue rectangle with text represent the plan/teamplan		The arrow with text represent an event posted within agent/team
	The arrow represents the uses of the JACK method establish to form task team, used in JACK Teams.		

Table 11: Sequence diagram notation

C.2 JACK Development Environment graphical notation

Table 12 describes the notation used in the JDE design diagrams shown in the report.

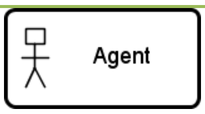
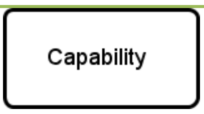
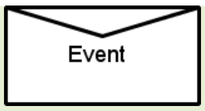
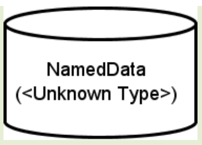
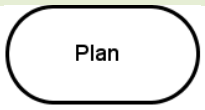
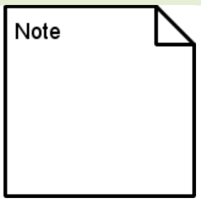
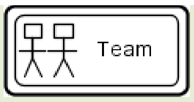
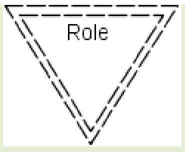
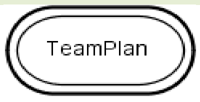
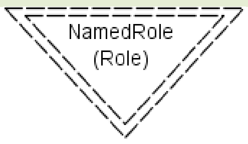
	Represent the JACK agent entity		*- Not used in this report-*
	Represent the JACK event entity		*- Not used in this report-*
	Represent the JACK plan entity		*- Not used in this report-*
	Represent the JACK team entity		Represent the *performs* JACK role entity declaration
	Represent the JACK teamplan entity		Represent the JACK role entity *required*

Table 12: JDE notation

Bibliography

1. **Einar Landre, Jørn Ølmheim, Geir Owe Wærsland, Harald Rønneberg Statoil ASA.** *Software Agents - An Emergent Software Technology That Enables Us To Build More Dynamic, Adaptable, and Robust System.* Texas : SPE Annual Technical Conference and Exhibition, September 2006.

2. **Jørn Ølmheim, Einar Landre, Eileen A. Quale StatoilHydro.** *Improving Production by use of Autonomous Systems.* Texas : SPE Annual Technical Conference and Exhibition, 2008.

3. **Spillum, Øystein.** *Delegation and coordination in autonomous oil production systems.* Trondheim : Norwegian University of Science and Technology, 2007.

4. **StatoilHydro.** StatoiHydro in brief. [Online] http://www.statoilhydro.com/en/AboutStatoilHydro/StatoilHydroInBrief/Pages/default.aspx.

5. **al., J. Tweedale et.** *Future Directions: Building a Decision Making Framework Using Agent.* s.l. : Springer-Verlag Berlin Heidelberg, 2008.

6. **J. Tweedale, N. Ichalkaranje, C. Sioutis, B. Jarvis, A. Consoli, G. Phillips-Wren.** *Innovations in multi-agent systems.* s.l. : Elsevier Ltd, 2006.

7. **(AOS), The Agent Oriented Software Group.** FAQ. [Online] [Cited: October 30, 2008.] http://www.agent-software.com/shared/products/faq.html#commsProtocols.

8. **Cheong, Christopher.** *A Comparison of JACK Intelligent Agents and the Open Agent Architecture.* Melbourne, AUSTRALIA : School of Computer Science and Information Technology, RMIT University.

9. **Wikipedia.** Cyber-physical system. [Online] http://en.wikipedia.org/wiki/Cyber-physical_system.

10. **Massachusetts), Bryan Horling (University of Massachusetts) and Victor Lesser (University of.** *A survey of multi-agent organizational paradigms.* s.l. : Cambridge University Press, 2005.

11. **Barber, K. Suzanne.** *Dynamic Adaptive Autonomy in Agent-based Systems.* s.l. : The University of Texas at Austin, 1999.

12. **Cheong, Christopher.** *An Empirical Investigation of Teamwork Infrastructure for Autonomous Agents.* Melbourne, AUSTRALIA : School of Computer Science and Information Technology, RMIT University.

13. **Alan F Hill, Fiona Cayzer, Peter R Wilkinson.** *Effective Operator Engagement with Variable Autonomy.* s.l. : 2nd SEAS DTC Technical Conference, 2007.

14. **Andrew Lucas, David Shepherdson.** *Architecture for Distributed Power Management for Autonomous Unmanned Vehicles.* s.l. : AOS Group, 2007.

15. **Vlassis, Nikos.** *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence.* s.l. : Department of Production Engineering and Management, Technical University of Crete, 2007.

16. **Angela Consoli, Jeffrey Tweedale and Lakhmi Jain.** The Link between Agent Coordination and Cooperation. *Intelligent Information Processing III.* s.l. : Springer Boston, 2007.

17. **Durfee, Edmund H.** *Scaling Up Agent Coordination Strategies.* s.l. : IEEE, 2001.

18. **Jinsong Leng, Colin Fyfe, and Lakhmi Jain.** *Teamwork and Simulation in Hybrid Cognitive Architecture.* s.l. : Springer-Verlag Berlin Heidelberg, 2006.

19. **Susannah Soon, Adrian Pearce, and Max Noble.** *A Teamwork Coordination Strategy Using Hierarchical Role Relationship Matching.* s.l. : Springer-Verlag Berlin Heidelberg, 2004.

20. **Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee & Jaron C. Collis.** *ZEUS: A Toolkit for Building Distributed Multi-Agent Systems.* s.l. : BT Laboratories, Martlesham Heath, 1999.

21. **H S Nwana, L Lee and N R Jennings.** *Co-ordination in software agent systems.* s.l. : BT Technol J Vol 14 No 4, 1996.

22. **Agent Oriented Software Pty Ltd.** *JACK™ Intelligent Agents Teams Manual.* s.l. : Agent Oriented Software Pty Ltd, 2005.

23. **Agent Oriented Software Pty. Ltd.** *JACK™ Intelligent Agents Agent Manual.* s.l. : Agent Oriented Software Pty. Ltd., 2005.

24. **Lise Engmo, Lene Hallen.** *Software agents applied in oil production, Master's thesis.* s.l. : Norwegian University of Science and Technology, 2006.

25. **J. Jarvis, R. Rönnquist, D. McFarlane, L. Jain.** *A team-based holonic approach to robotic assembly cell control.* s.l. : Elsevier Ltd., 2004.

26. **Sanjay Bisht, Aparna Malhotra, and S.B. Taneja.** Modelling and Simulation of Tactical Team Behaviour. *Defence Science Journal.* Vol. 57, No. 6, 2007 November.

27. **Daren, Yeo Huang-Yu.** *Automatic Protocol Generation Based on Commitment Machines.* s.l. : The University of Western Australia, 2004.

28. **Agent Oriented Software Pty. Ltd.** *JACK™ Intelligent Agents Development Environment Manual.* s.l. : Agent Oriented Software Pty. Ltd., 2005.

29. Eclipse (software). [Online] http://en.wikipedia.org/wiki/Java_eclipse.

30. **The Eclipse Foundation.** Eclipse Newcomers FAQ. [Online] http://www.eclipse.org/home/newcomers.php.

31. **Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslèn.** *Experimentation in Software Engineering: An Introduction.* s.l. : Kluwer Adademic Publishers, 2000.

32. **Mari Torgersrud Haug, Elin Marie Kristensen.** *Applicability and Identified Benefits of Agent Technology, master thesis.* s.l. : Norwegian University of Science and Technology, 2006.

33. **Oates, Briony J.** *Researching Information Systems and Computing.* s.l. : Sage Publications, 2006.

34. **Wikipedia.** Scalability. [Online] http://en.wikipedia.org/wiki/Scalability.

35. **Len Bass, Paul Clements, Rick Kazman.** *Software Architecture in Practice, Second edition.* s.l. : Pearson Education, Inc, 2003.

36. **Wikipedia.** High-level programming language. [Online] http://en.wikipedia.org/wiki/High-level_programming_language.

37. **Wooldridge, Michael.** *An introduction to MultiAgent Systems, 1.edition.* s.l. : John Wiley & Sons Ltd, 2002.

38. **Jovanovic, Gastón Eduardo Tagni and Dejan.** *Comparison of Multi-Agent Systems JACK vs 3APL.* s.l. : Departamento de Inform´atica, Universidade Nova de Lisboa.

39. **Agent Oriented Software Pty. Ltd.** *JACK™ Intelligent Agents Agent Practicals.* s.l. : Agent Oriented Software Pty. Ltd., 2005.

40. **Bevan Jarvis, Dan Corbett, and Lakhmi C. Jain.** *Beyond Trust: A Belief-Desire-Intention Model of Confidence in an Agent's Intentions.* s.l. : Springer-Verlag Berlin Heidelberg, 2005.

41. **Wikipedia.** Scalability. [Online] http://en.wikipedia.org/wiki/Maintainability.

42. **Agent Oriented Software Pty. Ltd.** *JACOB Manual.* s.l. : Agent Oriented Software Pty. Ltd., 2006.