Chapter 10

MULTI-AGENT WELL SCHEDULING: A PROTOTYPE IMPLEMENTATION USING CNP AND JADE

Jivko Hristov[1], Graham Lange[2], Fuhua Lin[1], M. Ali Akber Dewan[1], Xiaokun Zhang[1], Saadat Khan[1]
[1]*School of Computing and Information Systems, Athabasca University*
[2]*Encana Corporation, Canada*

Abstract: Efficient task allocation and resource scheduling have been challenging in oil and gas industries for many years, because they are influenced by a number of factors including resource availabilities, environment, regulations, stakeholders, finances, and market. With the advancement of information communication technologies (ICT), the oil and gas industries get the opportunities to increase production and to minimize operation costs through efficient resource management and task scheduling. This paper presents a prototype implementation of daily well scheduling using Java Agent Development Framework (JADE) — a multi-agent system platform. Coordination mechanism among the agents are implemented using the traditional Contract Net Protocol (CNP) which enables a flexible and efficient resource allocation leading to an intelligent management of the available resources and dynamic scheduling of the tasks across the well lifecycle. In the prototype model, sequence diagrams and class diagrams are used to show coordination mechanisms between different agents. Three different use cases initially demonstrate its effectiveness of the CNP-based coordination of multi-agent system approach to well scheduling. Future work on well scheduling in distributed and online environment is discussed.

Key words: Multi-agent systems, well scheduling, Contract Net protocol, JADE, business process modeling.

1. INTRODUCTION

Well scheduling is a highly dynamic in nature and complex problem in the oil and gas industries. Large oil and gas industries plan for their active drilling season in advance to utilize their equipment (or available resources) to the maximum and achieve high return on investments. Unfortunately, the active drilling season has limited window opportunity to complete all business goals. The work activities of the oil and gas industries largely depend on many internal and external factors, such availability of the resources, weather, regulations, and health and safety inspections. These factors are hard to manage by planners because it involves hundreds of processes and dependency tasks which change dynamically. This dynamic nature of the working environment warrants flexible technological solutions that will be able to allocate tasks and schedule resources intelligently to maximize industrial benefits.

A number of methods for well scheduling have been proposed in the last few years. To schedule the well activities, Hasle et al. (1995) developed a model by defining a particular well activities problem and generating a high quality and feasible schedule that can be inspected and modified by the user through interactive Gantt visualizers. In another work, Dimitrios et al. (2009) proposed a mixed integer nonlinear programming model. However, these models either require manual interaction or face challenges with the dynamic nature of the well scheduling. One possible solution would be the multi-agent systems, which can allow dynamic allocation of the tasks and schedule of the resources through agent negotiations, while managing all the related internal or external factors in a flexible manner. Multi-agent systems technologies have been widely used in complicated systems, which play a role in solving distributed and complex problems coordinately (Wooldridge & Jennings, 1995). In the multi-agent systems, agents can be software entities, computer programs or distinct objects in a larger software model that act autonomously on behalf of their users (Weiss, 1999; He et al., 2008). To resolve the dynamic scheduling problem we not only need agents but also need a way of effective communication between the agents and a way to negotiate terms and conditions to process the work.

Lange and Lin designed a system for well scheduling based on multi-agent systems platform (Lange & Lin, 2014). This system design is capable of negotiating among agents with agility and flexibility for a better solution of well scheduling. A prototype design and implementation of the above system using Java Agent Development Framework (JADE) and Contract Net Protocol (CNP) have been detailed in this paper. More specifically, “Add a well” among many different scenarios has been described. This scenario adds wells to the system and establishes communication to the service providers who can provide their bids. The tasks are assigned to the service providers who provide earliest completion time with minimal cost proposal. Percept sources, such as weather, roads condition, and health and safety have been considered. Three use cases are explained within the prototype implementation for performance analysis. The advantages of the proposed multi-agent system in “Add a well” scenario have been identified.

2. SYSTEM ARCHITECTURE

2.1 Overview

The well scheduling is a very complex and dynamic process with various tasks involved in commencing of a well. At a higher level, the tasks identified by the researchers are divided into five distinct phases: *Landman*, *Construction*, *Drilling*, *Completion*, and *Facilities*. The tasks are tightly coupled and can only be executed in sequential order. For example, the *Construction* task cannot be started before the *Landman* task is completed for a given well, or *Drilling* task cannot be begun before the *Construction* task is completed. The scheduling module needs to take this precedence information into account and expect to schedule all the tasks using an optimal plan. Each high level task is comprised of many lower level sub-tasks which must be completed before the overall process can move forward to the next milestone. A high level system diagram of the proposed business process model for well scheduling is shown in Figure 1, where several actors (or agents) are participating in the process of allocating tasks and scheduling resources for an optimal scheduling solution. It can also be observed that there is an actor representing each phase of the project like *Landman*, *Construction*, *Drilling*, *Completion*, and *Facilities*.

The well scheduling is comprised of many different scenarios such as “Add a well”, “Remove a well”, “Facilities Complete”, “Competition Finished”, “Drill Completion”, “Construction Complete” “Stop Work”, etc., each of which has many different functionalities. In this paper, the implementation of “Add Well” scenario with its functionalities has been described. The system determines the order of tasks for “Add Well” scenario as per predefined plan and the best scheduling time when the work can be completed. The scheduling decision is made based on previous service commitments thus the service providers will try to provide best available services in time. However, the work can be disrupted due to many external factors which stimulate a simple stop order. After completing the scheduling, the system is presented with the scheduled tasks which include the cost for every part of the whole project.

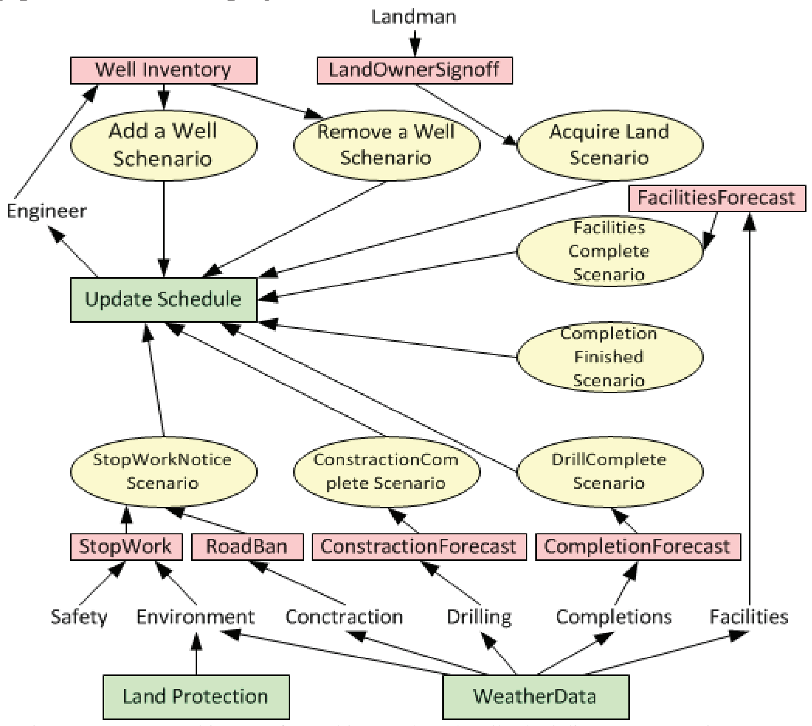


Figure 1: System architecture for multi-agent based well scheduling (Lange & Lin, 2014)

A high level sequence diagram is shown in Figure 2, which defines the high level message flow between agents. The process is initiated by the Engineer agent who adds a well to the system to be scheduled. The message is sent to the well agent who starts the negotiation process with the scheduling agents. There are five scheduling agents one for each high level phase – *Landman*, *Construction*, *Drilling*, *Completion* and *Facilities*. Each scheduling agent communicates with its respective service providers to find the best service offer which meets the shortest execution time utility. The utility function implemented by any of the agents can be developed to meet any business requirement. For this prototype the utility function will try to find the shortest path to complete the given work. Based on the high level system design, this paper defines the agents below with their respective functionalities. A high level sequence diagram of agents' interactions is shown in Figure 2.

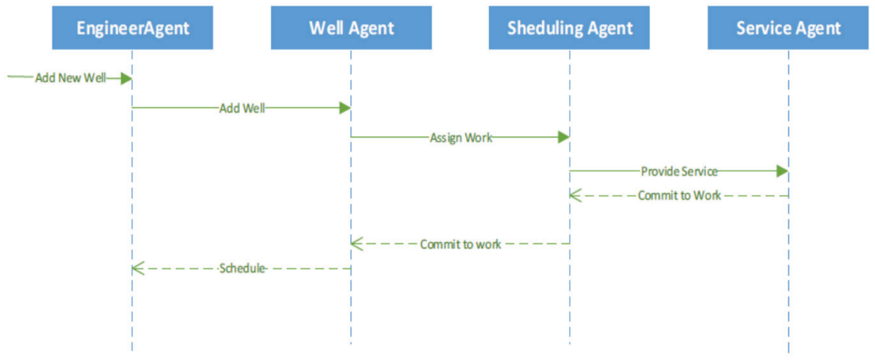


Figure 2: Sequence diagram of the multi-agent-based well scheduling system

2.2 Engineer Agent

This agent initiates the process of adding wells to the system to be scheduled. In the prototype, this functionality is simulated by implementing timer behavior which adds a new well in every 20 seconds to the maximum of nine wells. The upper limit was put in place to study the use cases defined in the research paper (Lange & Lin, 2014). If the limit is removed, the system will have no upper limit and continue to add wells after every 20 seconds for scheduling. The engineer agent executes three scenarios and for each scenario it adds three wells.

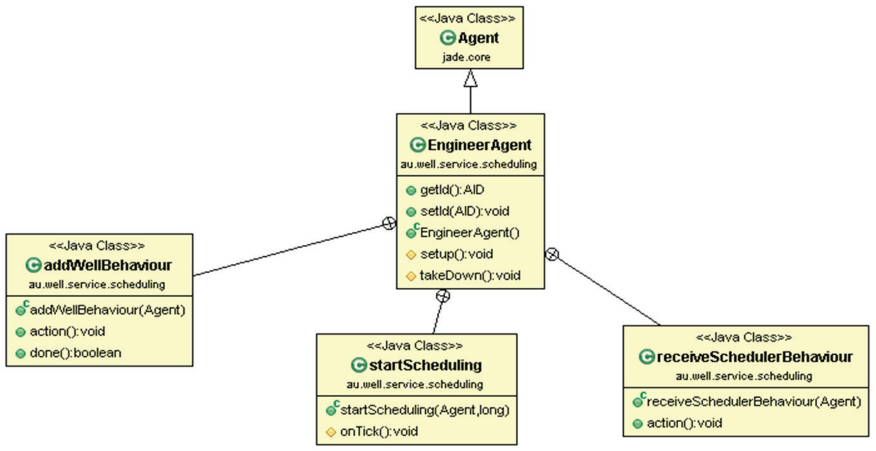
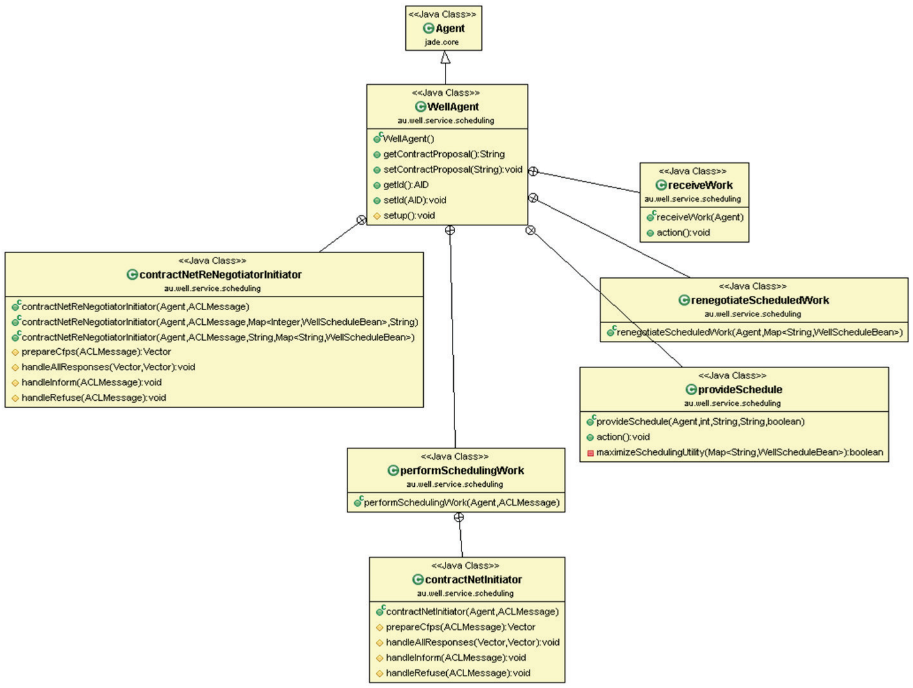


Figure 3: Class diagram for the engineer agent

The purpose of this design is to simulate the three different scheduling options the intelligent agents can use when scheduling the tasks. This scenario follows the close guidelines of the design in Lange and Lin's paper (Lange & Lin, 2014). Messages exchange by the agents is designed with the maximum flexibility to allow for future expansion of this application. Every time the agent wakes up it initiates the addWellBehaviour which send message to the well agent to add a new well to be scheduled. Class diagram of the engineer agent is shown in Figure 3.

2.3 Well Agent

The role of this agent is to accept requests from the engineer agent when a "Add a well" is initiated, to request for scheduling various tasks to the scheduling agent, and to send back the overall schedule and related cost to the engineer agent. In this design, the well agent plays the role of coordinator of the scheduling works. The agent extends JADE agent framework and implements several private classes to support the functionality as per well agent class diagram as shown in Figure 4.



<<Java Class>>
Agent
jade.core
<<Java Class>>
WellAgent
au.well.service.scheduling
WellAgent()
getContractProposal():String
setContractProposal(String):void
getId():AID
setId(AID):void
setup():void
<<Java Class>>
receiveWork
au.well.service.scheduling
receiveWork(Agent)
action():void
<<Java Class>>
contractNetReNegotiatorInitiator
au.well.service.scheduling
contractNetReNegotiatorInitiator(Agent,ACLMessage)
contractNetReNegotiatorInitiator(Agent,ACLMessage,Map<Integer,WellScheduleBean>,String)
contractNetReNegotiatorInitiator(Agent,ACLMessage,String,Map<String,WellScheduleBean>)
prepareCfps(ACLMessage):Vector
handleAllResponses(Vector,Vector):void
handleInform(ACLMessage):void
handleRefuse(ACLMessage):void
<<Java Class>>
renegotiateScheduledWork
au.well.service.scheduling
renegotiateScheduledWork(Agent,Map<String,WellScheduleBean>)
<<Java Class>>
provideSchedule
au.well.service.scheduling
provideSchedule(Agent,int,String,String,boolean)
action():void
maximizeSchedulingUtility(Map<String,WellScheduleBean>):boolean
<<Java Class>>
performSchedulingWork
au.well.service.scheduling
performSchedulingWork(Agent,ACLMessage)
<<Java Class>>
contractNetInitiator
au.well.service.scheduling
contractNetInitiator(Agent,ACLMessage)
prepareCfps(ACLMessage):Vector
handleAllResponses(Vector,Vector):void
handleInform(ACLMessage):void
handleRefuse(ACLMessage):void

Figure 4: Well agent class diagram

Once the scheduling request is received from the engineer agent, the well agent creates a sequential plan using SequentialBehaviour which defines the order of execution of scheduling work requests to the five scheduling agents. The sequential plan helps to control the sequence of execution of requesting schedules from the various scheduling agents when a new well is added to the system. Sub-behaviours are added to the plan by means of using addSubBehaviour() method which ensures that tasks are executed in the order they were added to the execution plan. The sequential behaviour can be instructed to terminate execution after the first child completes or after the last child finishes, it all depends on the business requirements. As previously described the order of scheduling work is sequential and the sequential behaviour helps enforce this rule and fire scheduling requests in the predefined order of execution (*Landman, Construction, Drilling, Completion and Facilities*).

Since the basic business requirements are that the schedules should not be overlapped, it implies that scheduling agents must share at the very minimum completion date with the next agent in the execution chain. Given that the well agent plays a coordinator role the well agent receives the schedule from every scheduling agent before it fires the request to the next one in the execution list which is controlled by the sequential behavior. The well agent uses the CNP to negotiate schedules with the respective scheduling agents. When the well work is fully scheduled the well agent uses a utility function to determine if the schedule of the works can be optimized. The optimization decision is based on a stop work order introduced in the schedule. The stop work order causes the well work schedule to be extended by the duration of the stop order thus costing the company additional cost.

The well agent ensures that the schedules returned to the engineer agent are optimal; therefore it requires the scheduling agents to re-plan the order of execution and find optimal work schedules across different wells (see sequence diagram in Figure 2). To support this functionality the solution introduces two new system operations – *Change* and *Move*. This new operations will request the scheduling agents to prioritize their work based on new provided start dates across the work they have already committed to. Alternatively more sophisticated implementation of this process could use iterated CNP which will be discussed further down this paper. The agent implements the business logic using different plans which drive out the decision of well scheduling.

The well agent recognizes this event by reviewing proposed schedules before returning them to the engineer agent. When reviewing schedules the well agent verifies schedules not only for the current well which is being created by it goes back to the previous schedules to find potential work stop gaps which can be utilized to optimize current and previously committed schedules to reduce work duration. The scheduling agent will accept the optimization request and assess the available options.

2.4 Scheduling Agents

The application design introduces a scheduling agent for every task in the process flow, for example one scheduling agent for *Landman*, another one for *Construction* etc., as per the scheduling agent class diagram as shown in Figure 5. Since the application has many scheduling agents, the design abstracts the common Contract Net protocol framework to the abstract class SchedulerAgent so every agent can implement the abstract methods and reuse the common functionality of the base class.

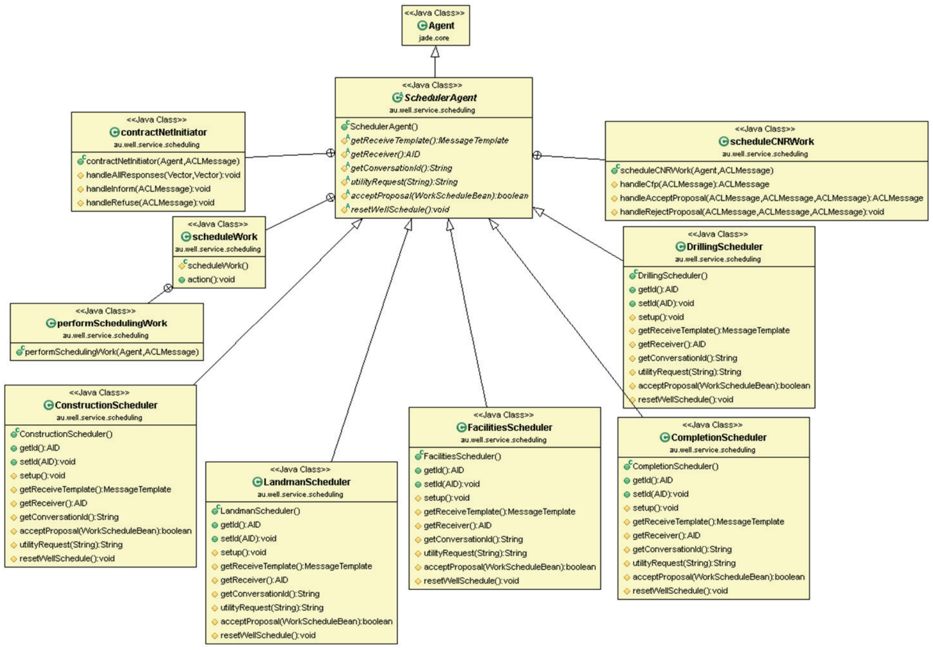

<<Java Class>>
Agent
jade.core
<<Java Class>>
SchedulerAgent
au.well.service.scheduling
SchedulerAgent()
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
utilityRequest(String):String
acceptProposal(WorkScheduleBean):boolean
resetWellSchedule():void
<<Java Class>>
contractNetInitiator
au.well.service.scheduling
contractNetInitiator(Agent,ACLMessage)
handleAllResponses(Vector,Vector):void
handleInform(ACLMessage):void
handleRefuse(ACLMessage):void
<<Java Class>>
scheduleCNRWork
au.well.service.scheduling
scheduleCNRWork(Agent,ACLMessage)
handleCfp(ACLMessage):ACLMessage
handleAcceptProposal(ACLMessage,ACLMessage,ACLMessage):ACLMessage
handleRejectProposal(ACLMessage,ACLMessage,ACLMessage):void
<<Java Class>>
scheduleWork
au.well.service.scheduling
scheduleWork()
action():void
<<Java Class>>
performSchedulingWork
au.well.service.scheduling
performSchedulingWork(Agent,ACLMessage)
<<Java Class>>
DrillingScheduler
au.well.service.scheduling
DrillingScheduler()
getId():AID
setId(AID):void
setup():void
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
utilityRequest(String):String
acceptProposal(WorkScheduleBean):boolean
resetWellSchedule():void
<<Java Class>>
ConstructionScheduler
au.well.service.scheduling
ConstructionScheduler()
getId():AID
setId(AID):void
setup():void
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
acceptProposal(WorkScheduleBean):boolean
utilityRequest(String):String
resetWellSchedule():void
<<Java Class>>
LandmanScheduler
au.well.service.scheduling
LandmanScheduler()
getId():AID
setId(AID):void
setup():void
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
utilityRequest(String):String
acceptProposal(WorkScheduleBean):boolean
resetWellSchedule():void
<<Java Class>>
FacilitiesScheduler
au.well.service.scheduling
FacilitiesScheduler()
getId():AID
setId(AID):void
setup():void
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
utilityRequest(String):String
acceptProposal(WorkScheduleBean):boolean
resetWellSchedule():void
<<Java Class>>
CompletionScheduler
au.well.service.scheduling
CompletionScheduler()
getId():AID
setId(AID):void
setup():void
getReceiveTemplate():MessageTemplate
getReceiver():AID
getConversationId():String
utilityRequest(String):String
acceptProposal(WorkScheduleBean):boolean
resetWellSchedule():void

Figure 5: Scheduling agent class diagram

The base class will call each abstract method which will be implemented by all five scheduling agents. Each individual agent will know the services providers and maintain internal work schedule. The scheduling agent will support three operations like *AddWell*, *Change* and *Move*. The first operation, *AddWell*, will support adding new well to the schedule. Before any new work is taken on the scheduling agent executes its utility function to find best offer it needs to meet the requested demand. This proposal is sent to the service agent for commitment or counteroffer. As part of the contract net negotiation protocol when the scheduling agent receives a proposal from the servicing agent it validates all proposal to find the best offer. Each scheduling agent can contact one to many service providers with work request which they can bid on. When all bids are received as part of the contract net negotiation process the scheduling agent reviews all offers and selects the best offer, the offer that is either matching the proposed start date or the one with shortest duration. The scheduling agent declines all offers with the exception of the one it accepts for which it requests the service agent to commit to it.

The second operation *Change* will indicate to the scheduling agent to consider rescheduling work for various wells based on the proposed start dates. The agent will validate its internal committed work and decide if optimizations are possible. If optimization is possible the agent will engage the service provider to arrange for new commitments. The scheduling agent always verifies the offers coming back from the service providers against their internal records. If the service provider offer is different from the recommended schedule which the scheduling agent provided upon submitting the request for work messages to all agents it either can decline it or accept the best offer. When the best offer is accepted and committed to by the service provider the scheduling agent records it in its internal memory. Irrespective of the operation executed the underlying negotiation protocol used is the same. All interactions between agents when negotiation is required use the CNP framework.

2.5 Service Agents

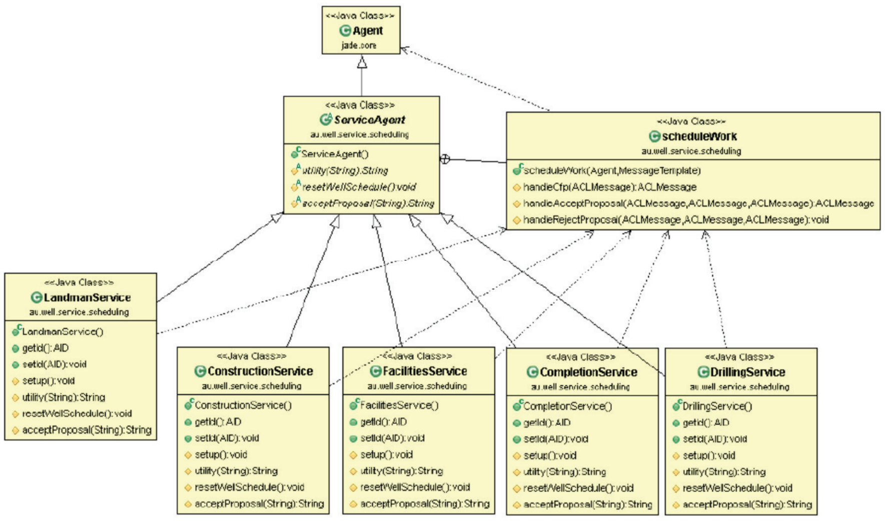


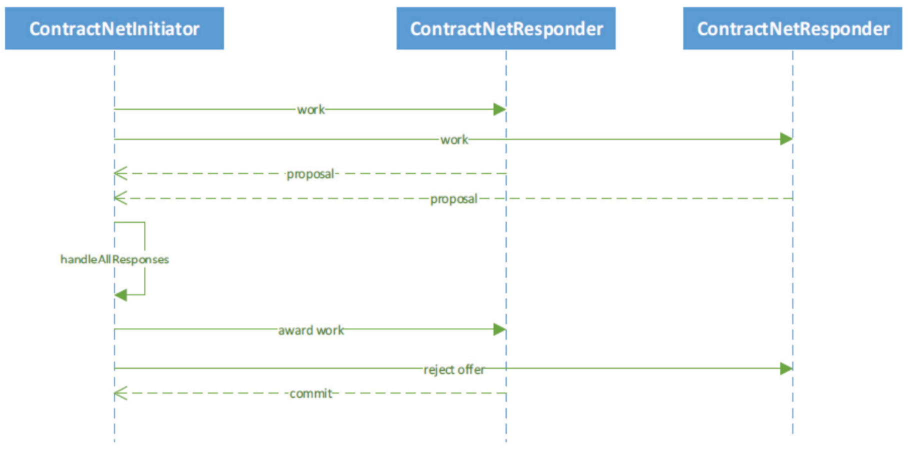
Figure 6: Service agent sequence diagram

This agent represents the service providers in the system. The service agents' base class (ServiceAgent) implements all common service functionality such as the contract net responder and provides several abstract methods which are implemented by the service providers. Each service provider as per Services Class Diagram as shown in Figure 6 implements its own utility functions and corresponding abstract methods. Services agents bid on proposed work by the scheduling agents (see sequence diagram in Figure 2). The service agents only implement Contract Net responder functionality because the services only bid on work and either their bid is accepted or rejected.

3. PROTOTYPE IMPLEMENTATION IN JADE FRAMEWORK

This section describes the implementation details of the Contract Net protocol in JADE framework. The implemented prototype extensively uses the Contract Net protocol to exchange messages between agents which use the protocol to negotiate for best solutions by optimizing their internal goals.

In CNP for agent negotiation, the process (as Figure 7 shows) starts with contractor agent broadcasting works to service provider agents. The service provider agents send their proposals to the contractor agent. The contractor agent looks at all proposals and finds the best proposal that maximizes its utility function. Once the optimal proposal is found, the contractor agent sends accept message to the agent that provided the best offer and reject message to the other agents. The agent that receives the accept message needs to commit to the offer by sending a commit message to the contractor.



Contract Net Protocol Diagram

Figure 7: Sequence diagram of Contract Net protocol.

In the prototype implementation, the well agent is used as a coordinator which is responsible for providing requirements from one scheduler agent to the next aiming to achieve its goal of completing scheduled well work in the shortest time. The scheduling agent receives its work from the well agent with proposed start time of the work. Every scheduling agent tries to achieve this target by requesting the service providers match the proposed timelines. The benefit with this architecture is that the scheduling agents don't know of each other's existence. They are decoupled from one another and work independently. Another positive side for this architecture is that if additional work or new phase is introduced in the well work schedule, it can easily be added to the execution plan of the well agent and none of the scheduling agents will need to be concerned with it nor there will be any changes for those scheduling agents.

Alternatively this solution can be built with different architecture where well agent initiates the work by broad casting it to the agents. In this model the agents can be chained and each agent knows which other agent it can receive work from and once completed who to pass the work to. This model is obviously a bit more complicated and harder to manage due to fact that agents will need to have previous knowledge about their surrounding agents in the chain of work.

Another observation about the agent platform is that when agents fail they stop receiving messages and become unresponsive which may be an issue if this platform is used to implement enterprise solutions because the system needs to be stable and reliable. This fining will require the application to be well designed and developed to handle and deal with any unforeseen exceptions during application execution to avoid the agent to become unresponsive.

Scalability of the platform will need to be further assessed. The business logic in each method of the behavior is executed only after the previous method exits. This constraint will require very careful application design so that behaviors are executed efficiently. The behaviors should not be developed with blocking logic because if the business logic requires other work to be executed while waiting it will never be executed since the blocking behavior method will never exit to allow the scheduler to invoke the next behavior work in the pipeline. The reason for this constrain is because agents are designed to execute in environments will limited resources thus this architecture definitely suites systems in that space. However, in larger environments where this constrain is not applicable will see this requirement as a negative constrain.

Sharing data between different behaviors is more challenging than in regular applications. The reason for it is because behaviors are preconstructed before they are executed thus information that is obtained by the first behavior via some business functionality can't be passed to the next behavior in the chain of execution because it was not available when the plan was created. The way this problem is resolved in this environment is by either using local variables in the agent or its parents (Bellifemine et al., 2007). However, keeping data stored in the agents or parents variables is not a best practice because it prevents common logic to be reused. Therefore, it is best if data is externalized from the agents and stored in Data Store class included in the jade framework.

JADE platform executes logic in a single thread using the concept of behaviors. The behaviors provide several different implementations that the developers can use to solve various business requirements. This project utilizes several of the JADE behaviors to solve this assignment design requirements. The sequential behavior pattern is used to execute tasks in specific order to ensure that no one task is executed before its precedent. Well agent is perfect example where this behavior is used. As per the business requirements the first agent needs to receive the commitments of the first service provider before it can provide requirements to next service provider in the chain of execution. This pattern is heavily utilized to enforce sequential execution of tasks. Another behavior used is one shot behavior. This framework is used in event when specific task is required to be executed just once. Typical use case for using it is sending a message to another agent without requiring waiting for the response. Another interesting functionality implemented using behavior framework is ticker behavior. It gives the application capability to execute behaviors within preprogrammed intervals. Very important behavior called cyclical is used to constantly listen for arriving messages. Since the agents constantly exchange this behavior can help implement asynchronous message exchange pattern where agent can send a message to other agents and not necessarily wait for any response immediately. This behavior can help implement asynchronous message pattern exchange between agents.

4. CASE STUDIES

4.1 Use Case 1

This use case depicts the three well schedules in which there are no external forces that impact the committed work timelines by the various parties. As outlined each phase takes exactly the allowed time to complete before the next phase can begin. The test case implemented confirmed Lange and Lin (Lange & Lin, 2014) estimate that if the work is completed as per requirements the scheduling of the three well test solution is optimal. The work duration for this small project completes in 12 months.

Tasks	**Well #1**	**Well #1**	**Well #1**
Land	$50,000	$50,000	$50,000
Construction	$100,000	$100,000	$100,000
Drill	$2,000,000	$2,000,000	$2,000,000
Complete	$1,000,000	$1,000,000	$1,000,000
Facilities	$500,000	$500,000	$500,000
Total	**$3,650,000**	**$4,650,000**	**$3,650,000**
			$10,950,000
	Total Time		**13 months**

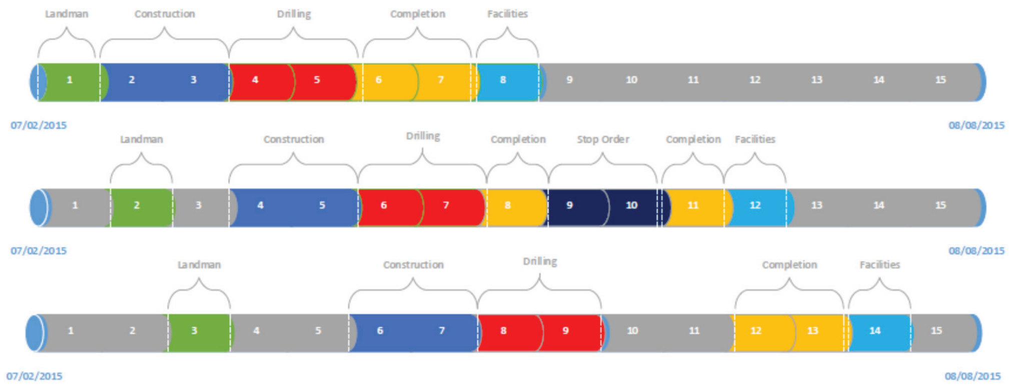


Figure 8: Scheduling result for Case 1

4.2 Use Case 2

Use case two implementations execute the same model where three wells will need to be scheduled. The difference here is that stop order is introduced in the middle of the completion phase for 2 months. Naturally this will push out the completion phase by two months resulting in delays for the overall schedule which not only impacts well two schedule but also well three schedule. Well three's schedule is also delayed because the completion work can't commence as planned. This stop order directly impacts the optimal schedule which results in 2 months delay. Therefore the entire duration of this project would be equivalent to 14 months.

Tasks	**Well #1**	**Well #1**	**Well #1**

Land	$50,000	$50,000	$50,000
Construction	$100,000	$100,000	$100,000
Drill	$2,000,000	$2.000,000	$2,000,000
Complete	$1,000,000	$1,300,000	$1,000,000
Standby		$400,000	
Facilities	$500,000	$500,000	$500,000
Total	**$3,650,000**	**$4,350,000**	**$500,000**
			$11,650,000
		Total Time	**17 months**

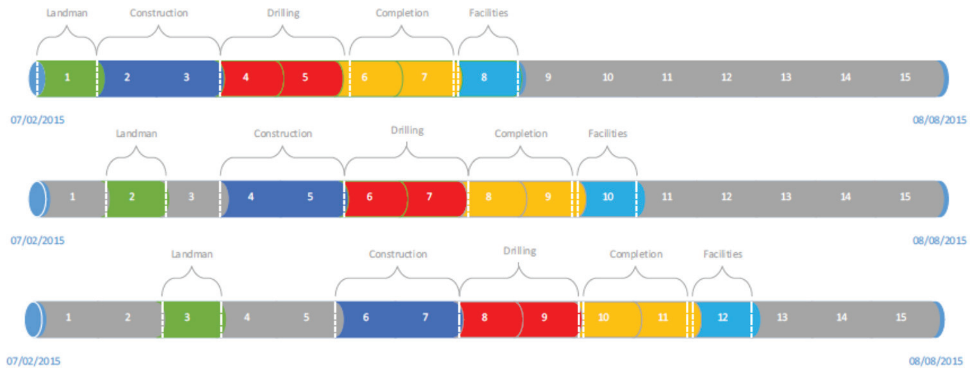


Figure 9: Scheduling result for Case 2

4.3 Use Case 3

In this use case the same stop order is introduced into the second well completion phase schedule however the agents consider optimizing the schedule when stop order is introduced in any phase of the well schedule. The optimization reduces the duration of the project by two months helping the agents achieve their overall goal.

Tasks	**Well #1**	**Well #2**	**Well #1**
Land	$50,000	$50,000	$50,000
Construction	$100,000	$100,000	$100,000
Drill	$2,000,000	$2,000,000	$2,000,000
Complete	$1,000,000	$1,300,000	$1,000,000
Standby		$200,000	
Facilities	$500,000	$500,000	$500,000
Total	**$3,650,000**	**$3,950,000**	**$3,650,000**
			$11,450,000

		Total Time	**15 months**

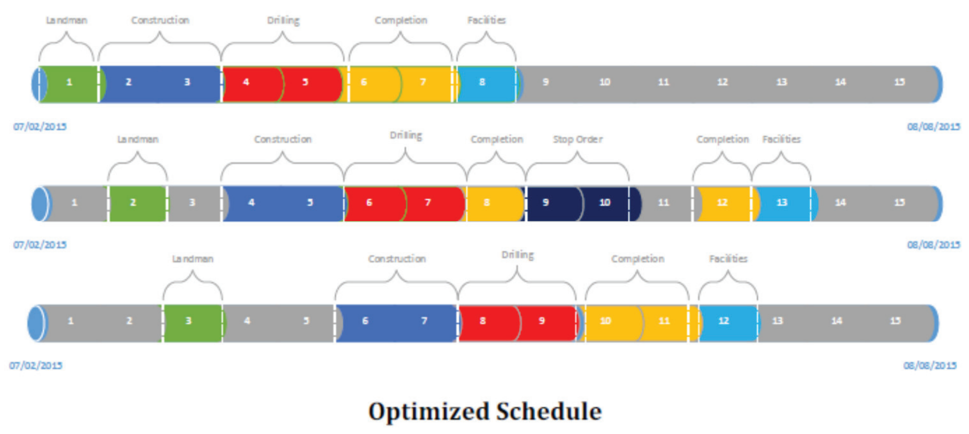


Figure 10: Scheduling result for Case 3

5. CONCLUSION

Multi-agent based well scheduling approach based on the notion of virtual enterprise in the oil and gas industry can solve a number of problems that conventional approaches cannot solve. By implementing a multi-agent system to well scheduling we can automate this manual effort. Multi-agent systems approaches are well suited for a dynamic environment and by working through the design it is evident that the approach will work. The multi-agent approaches allows for a certain amount of flexibility and timeliness not provided in traditional systems.

Coordination of CNP with different agents was used as the negotiation mechanism for solving the resource and scheduling problem. This should be a suitable approach to solving the problem of negotiating multiple resources across the schedule. However, the limitation of the current implementation is that it is one task version of CNP. There are weaknesses of this method: lack of optimality. This lack of optimality is due to decisions that lead to the myopic behavior of decision-making entities. For example, the case of a single task in reactive scheduling, but this is actually a degenerate case of the CNP. That is, the agents are ignorant in temporal aspect without taking into account what the other proposals that may arrive later. The temporal aspect comes from the fact that the bidders are aware only of the requests for bids already received, and not of those that are on the point of arriving. An agent may commit it to carry out a task for which it is not really very well suited, and therefore miss contracts that would correspond better to its true qualities (van Parunak, 1987). Therefore, future research could include additional negotiation approaches such as Combinational Auctions (Sandholm, 2002).

What is required to further this work is to further implement the solution and run a large number of tests to ensure that the results are acceptable to validate and verify the model. Since the benefits of applying multi-agent approach to the well scheduling problem is directly proportional to the size of the overall well program, the tests should include a test for scalability. Tests should be conducted for 50,100, and 1000 well programs. The results should be compared to traditional systems currently in use. These systems include manual scheduling, Microsoft Project and Oracles Primavera. The comparisons would further support the implementation of a multi-agent approach in an enterprise environment. In addition to comparing the system to scheduling and planning tools, further research into comparison to a traditional BPM method would help to understand which is a better application. Further research on the coordination of multiple Contract Net protocols in a system should be conducted. This would help to determine whether this approach is the most applicable for the well scheduling problem.

REFERENCES

Arash Mousavi, A., Nordin, M., and Ali Othman, Z. (2011, April 19). Ontology-driven coordination model for multiagent-based mobile workforce brokering systems. *Applied Intelligence, 36*(4), 768-787. doi:10.1007/s10489-011-0294-z

Bellifemine, F., Caire, G., and Greenwood, D., (2007). Developing multi-agent systems with JADE, Wiley

Gerogiorgis, D., Kosmidis, V., and Pistikopoulos, E. (2009). Mixed Integer Optimization in Well Scheduling. *Encyclopedia of Optimization*, pp. 2247-2270. doi:10.1007/978-0-387-74759-0_395

Hasle, G., Haut, R., Johansen, B., & Ølberg, T. (1995). Well activity scheduling-an application of constraint reasoning. *Artificial Intelligence in the Petroleum Industry: Symbolic and Computational Applications II*, 209-228.

He, L., Liu, Y.-X., Xie , H.-l., and Zhang, Y. (2008). Job shop dynamic scheduling model based on multi-agent," in Control and Decision Conference, Yantai, Shandong, China. 829 - 833

Lange, G., & Lin, F. (2014). Modeling Well Scheduling as a Virtual Enterprise With Intelligent Agents. Proceedings of *IEEE*

Computational Science and Engineering Conference. Chengdu, China. 89-96

Parunak, H. V. D. (1987). Manufacturing Experience with the Contract Net. In Distributed Artificial Intelligence, M. Huhns (Ed.), Pitman.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artif. Intell.* 135, 1-2, 1-54.

Weiss, G. (1999). *Multiagent Systems A Modern Approach to Distributed Modern Approach to Artificial Intelligence.* MIT Press.

Wooldridge, M., and Jennings, N. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review, 10*, 115-152.

ACKNOWLEDGEMENTS

Here we thank Natural Sciences and Engineering Research Council (NSERC) of Canada and Encana Services Inc. (Canada) for its financial support to the research project.