Planning and Scheduling Ship Operations on Petroleum Ports and Platforms

Tiago Stegun Vaquero[1] and **Gustavo Costa**[2] and **Flavio Tonidandel**[3]
Haroldo Igreja[4] and **José Reinaldo Silva**[2] and **J. Christopher Beck**[1]
[1]Department of Mechanical & Industrial Engineering, University of Toronto, Canada
[2]Department of Mechatronics Engineering, University of São Paulo, Brazil
[3]IAAA Lab, University Center of FEI - São Bernardo do Campo, Brazil
[4]Transpetro, PETROBRAS, Brazil
{tvaquero,jcb}@mie.utoronto.ca, {gustavorochacosta, reinaldo}@usp.br, flaviot@fei.edu.br, higreja@petrobras.com.br

Abstract

In this paper, we address the process of modeling planning and scheduling ship operations on petroleum platforms and ports. The general problem to be solved is based on the transportation and delivery of a list of requested cargo to different locations considering a number of constraints and elements based on a real problem of Petrobras – the Brazilian Petroleum Company. The objective is to optimize a set of costs brought by the execution of a schedule. Modeling the problem in UML and then translating to PDDL is shown to be feasible and practical by using itSIMPLE. However, although domain-independent planners can provide valid solutions to simplified versions of the problem, they struggle with a more realistic version.

Introduction

With the discovery of a promising massive oilfield beneath 2000 to 3000 meters of water in 2007, the Brazilian government has been investing in advanced technologies and infrastructure for deep water extraction of oil and natural gas. New discoveries in what is called the *pre-salt* basin created even more challenges in deep water exploitation and in several underlying engineering problems in order to make this effort secure, profitable and safe for the environment. One of the challenges is the planning and scheduling of vessels which transport goods, components and tools between crowded ports on land to platforms in the ocean. The supply of these elements to the network of platforms is essential to maintaining a fully operational oil extraction station off the Brazilian coast. Potential expansion of the number of platforms must be carefully studied and optimized to result in minimal impact on the environment. Hence, studying the planning and scheduling of ship operations in those ports and platforms is one of the aims of Petrobras.

The general problem to be solved is based on the transportation and delivery of a list of requested cargo to different locations considering a number of constraints and elements such as available ports, platforms, vessel capacity, weights of cargo items, fuel consumption, available refueling stations in the ocean, different duration of operations, and costs. Given a set of cargo items, the problem is to find a feasible plan that guarantees their delivery while respecting the constraints and requirements of the ship capacities. The objective is to minimize the total amount of fuel used, the size of waiting queues in ports, the number of ships used, the makespan of the schedule and the docking cost. The problem has a number of features that have been addressed by heuristic-based space-state search. Thus, it is a realistic problem that may be amenable to planning technology.

Since the 1980s there has been a recurring discussion in the literature regarding the relationship between Artificial Intelligence (AI) planning and optimization problems. A particular contrast is the traditional satisfaction-oriented bias of AI planning (Kautz and Walser 1999; 2000) versus the substantial focus and exploitation of cost functions in optimization approaches studied in Operations Research. Developing solvers for planning & scheduling (P&S) applications that demand both satisfaction-oriented approaches and optimization mechanisms is, with the current technology, still challenging. This is also a challenge faced by Knowledge Engineering (KE) tools and approaches: how to allow designers (both problem-domain experts and planning experts) to model problems requiring sophisticated planning capabilities, reasoning about time constraints, and the expression and minimization of complicated cost functions. There are not many KE tools available for modeling these sorts of problems in the AI P&S literature (Vaquero, Silva, and Beck 2011). As a consequence, the problem presented in this paper is one of the challenge domains in the Fourth International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012).

In this paper, we describe the modeling process we undertook using an AI P&S approach to study one potential expansion of the network of platforms. Our aim is (1) to investigate and describe the modeling process of the ship operation problem in such a network utilizing KE tools (in this case, the itSIMPLE tool (Vaquero et al. 2009)) and standard languages from AI P&S (e.g., PDDL), and (2) to study the use of available domain-independent planners and their performance in solving the model and generating plans. Even though we do not use real data in this paper due to privacy policies, it does not change or reduce the challenge of modeling and solving the problem. The main contributions of this work are: the design of a knowledge model for the planning and scheduling problem of ship operations in petroleum ports and platforms following the AI P&S approach; and experimental studies that explore the performance of domain-independent, heuristic-based planners on a realistic P&S problem that includes numeric variables and time constraints.

This paper is structured as follows. Firstly, we describe the problem, its restrictions and requirements. Secondly, we describe the design process, focusing on the modeling approach using itSIMPLE. Next, we provide experimental results obtained by selected domain-independent planners when solving problem instances of increasing size in two different scenarios: with and without time constraints. We conclude with a discussion of the results.

Problem Description

The problem of planning and scheduling ship operations on petroleum platforms and ports includes vessel capacity restrictions, the optimization of multiple, coupled objectives, and many others features that make this domain a challenge to AI planning systems. The model of this problem was simplified to focus on the need to provide transportation of goods from ports on the land to platforms in the ocean. In this problem, we consider two strips of the Brazilian coast: *Rio de Janeiro* and *Santos*. Each strip has one port (port *P1* at Rio de Janeiro and port *P2* at Santos) where the loading activities of cargo items occur to support petroleum extraction in deep water.

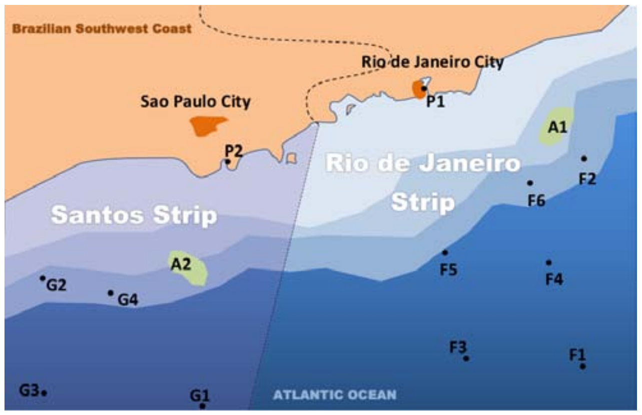


Figure 1: Layout of the strips and position of the ports on the Brazilian Coast.

Both strips contain a set of ocean platforms: six platforms (*F1, . . . , F6*) in the Rio de Janeiro strip and four (*G1, . . . , G4*) in the Santos strip. The ports are located 200 km from each other while platforms are located from 100 km to 300 km from ports. These platforms frequently require cargo that must be delivered from a port to the requesting platform. Each group of platforms is located in the strip connected to their respective port onshore, as shown in Figure 1. A vessel loads cargo at a port (and sometimes at platforms) and travels to target points for delivery of part or all of its cargo. After completing a delivery, ships go to the waiting areas off-shore. There is one waiting area in each strip: the one in Rio de Janeiro (called *A1*) is located 120 km (radial distance) from port *P1* and the one in Santos (called *A2*) is located 100 km from port *P2*. The distance between *A1* and *A2* is 340 km. Tables 1, 2 and 3 provide the distances between ports, platforms and waiting areas in this problem, as illustrated in Figure 1.

	P1	F2	F3	F4	F5	F6
F1	300km	168km	168km	120km	260km	240km
F2	160km	-	240km	120km	168km	120km
F3	280km	240km	-	120km	168km	260km
F4	200km	120km	120km	-	120km	168km
F5	160km	168km	168km	120km	-	120km
F6	130km	120km	260km	168km	120km	-

Table 1: Distance between platforms and ports in the Rio de Janeiro strip.

	P2	G2	G3	G4
G1	300km	200km	120km	260km
G2	180km	-	260km	120km
G3	280km	260km	-	200km
G4	140km	120km	200km	-

Table 2: Distance between platforms and ports in the Santos strip.

Vessels are the main resource used to transport cargo items from/to ports and platforms. A set of ships is responsible for supplying the platforms. In this problem, we consider ten available vessels (*S1, . . . , S10*): six of them have the Rio de Janeiro strip as their base and four of them have Santos as base. Cargo items (*C1, . . . ,CN*) refer to products, food, equipment, and parts that must be delivered to platforms and/or ports. They are represented as containers in this work.

Given a set of cargo items to transport and their respective locations, the challenge is to find a feasible plan that delivers all cargo properly, minimizing the total amount of fuel used, the makespan and the costs involved. Such a feasible plan must respect the requirements described in the remainder of this section.

Ports and Platforms: The ports can dock two ships simultaneously for loading, unloading and refueling. After receiving two ships, all further requests for docking have to be queued. The cost for docking is is 1000 Brazilian Reais per hour. This cost is applied only when the vessel is moored in a port, and is computed from the time the vessel starts docking to the time it undocks. We do not address the packing and organization of the cargo in the vessel, only the loading/unloading rate.

Besides the port, a vessel can refuel at a subset of the platforms. For this problem, we consider platforms *F5* and *G3* as capable for providing refueling operations. The refueling operation of a vessel is performed at a rate of 100 liters per hour in both ports and platforms. Only one vessel can dock to a platform at any given time.

Vessels: Each ship has a limited capacity for cargo items (100 tons) and a limited fuel tank (600 liters). Traveling with the specified speed average of 70 km/h, ships consume

	F1	F2	F3	F4	F5	F6	A1	A2	P1	P2
G1	468km	580km	420km	500km	380km	520km	540km	320km	350km	300km
G2	580km	468km	380km	520km	300km	500km	540km	110km	400km	180km
G3	588km	600km	420km	560km	580km	580km	580km	400km	450km	280km
G4	600km	588km	580km	580km	420km	580km	570km	180km	420km	140km
A1	200km	40km	320km	280km	180km	80km	-	340km	120km	270km
A2	340km	380km	370km	340km	280km	300km	340km	-	270km	100km
P1	300km	160km	280km	200km	160km	130km	120km	270km	-	200km
P2	380km	290km	320km	340km	270km	300km	270km	100km	200km	-

Table 3: Distance between the platforms, ports and waiting areas in the Rio de Janeiro and Santos strips.

1 liter of fuel each 3 km when traveling fully loaded and 1 liter each 5 km if empty. We assume that all ships have the same capacity for cargo and the same average speed.

Before executing any activity in a port or a platform, ships must perform a docking process. The docking or undocking process of a vessel at a port takes 1 hour, whereas at a platform it takes 0.5 hour. Ships can be docked at ports and platforms to load and unload cargo items, to be refueled, or both. The loading and unloading processes can be done either at the platforms in the ocean or at the port onshore; however, they cannot be done at the same time in a given location. Each vessel can perform the loading/unloading operation with a rate of 1 ton per hour. Refueling can be done at the port or at platforms that have a refueling system, and can be performed during loading or unloading. The rates for refueling are the following: 100 liters per hour at a platform; 100 liters in half an hour at the ports.

Cargo Items: Cargo items can be carried by ships from one location to another, and we disregard the order of loading and unloading in this problem. Since each cargo item has a specified weight, loading a ship is limited by the capacity of that ship. The weight for each cargo item is specified in the request and is considered input data for the problem.

Waiting areas in the ocean: All vessels have to be in a waiting area at the beginning its multiple deliveries. At the end of all deliveries the vessels must go back to a waiting area to wait for the next requests. It is possible to send a vessel located initially in one waiting area to another waiting area of the other strip. However, it is important to have a balanced number of vessels in each one. The ideal balance is 6 vessels in the Rio de Janeiro area *A1* and 4 in the Santos waiting area *A2*.

The use of waiting areas is important to avoid long and unnecessary docking periods at the ports (since there is a cost associated with each docking period) and at the platforms. When parking at the waiting areas, ships must have sufficient fuel to return to a refueling location.

The Modeling Process with itSIMPLE

The KE tool called itSIMPLE (Vaquero et al. 2007; 2009) was used to support the construction and development of the domain model for the problem described above. itSIMPLE's integrated environment focuses on the crucial initial phases of a design.

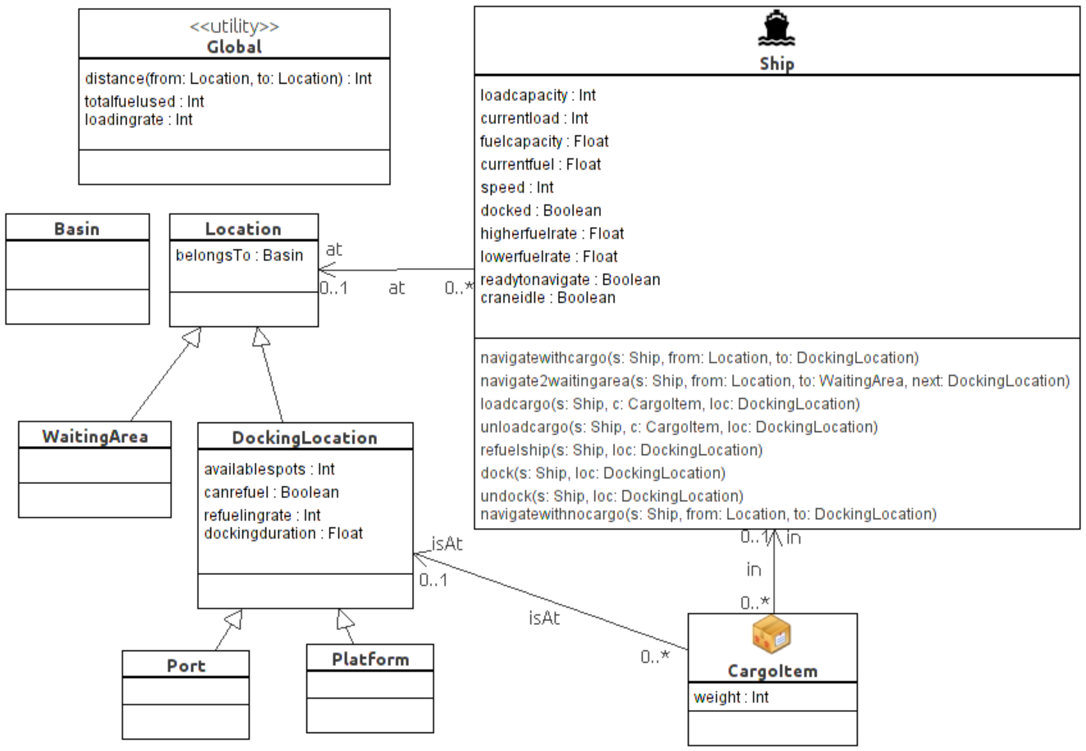
The tool allows users to follow a disciplined design process to create knowledge intensive models of planning domains, from the informality or semi-informality of real world requirements to formal specifications and domain models that can be read by domain-independent planners (those that read PDDL). The suggested design process for building planning domain models includes the following phases: requirements specification; modeling; model analysis; testing with planners; and plan evaluation (Vaquero et al. 2007). These phases are inherited from Software Engineering and Design Engineering, combined with real planning domain modeling experiences. In this paper, we focus on three of the main phases of such a design process: modeling, testing with planners, and plan analysis.

Domain Modeling

Modeling in itSIMPLE follows an object-oriented approach. Requirements are gathered and modeled using *Unified Modeling Language* (UML) (OMG 2005), a general purpose language broadly accepted in Software Engineering and Requirements Engineering. UML is used to specify, visualize, modify, construct and document domains or artifacts, generally following an object-oriented approach. The tool allows the modeling of a planning problem using diagrams such as *class diagram*, *state machine diagram*, *timing diagram*, and *object diagram*

The class diagram represents the static structure of the planning domain. It shows the existing types of objects, their relationships, properties, operators (actions) and constraints. Class attributes and associations give a visual notion of the semantics of the model. Figure 2 shows the class diagram designed for the Petrobras problem. The diagram consists of nine classes: *Basin*, *Location*, *WaitingArea* (a specialization of Location), *DockingLocation* (also a specialization of Location), *Port* (a specialization of *DockingLocation*), *Platform* (also a specialization of *DockingLocation*), *Cargo*, *Ship*, and *Global* (the class *Global* is a utility class that stores global variables that are accessed from all other classes). The classes illustrated in Figure 2 model all the entities relevant to the problem.

The class *Ship* has several properties that match the requirements. We tried to use straightforward names for these properties to facilitate the understanding of the model and provide an intuitive semantics for a non-planning expert (e.g., *loadcapacity* and *currentload* are numeric values representing the capacity of the ship and its current load); how-



<<utility>>
Global
distance(from: Location, to: Location) : Int
totalfuelused : Int
loadingrate : Int
Ship
loadcapacity : Int
currentload : Int
fuelcapacity : Float
currentfuel : Float
speed : Int
docked : Boolean
higherfuelrate : Float
lowerfuelrate : Float
readytonavigate : Boolean
craneidle : Boolean
navigatewithcargo(s: Ship, from: Location, to: DockingLocation)
navigate2waitingarea(s: Ship, from: Location, to: WaitingArea, next: DockingLocation)
loadcargo(s: Ship, c: CargoItem, loc: DockingLocation)
unloadcargo(s: Ship, c: CargoItem, loc: DockingLocation)
refuelship(s: Ship, loc: DockingLocation)
dock(s: Ship, loc: DockingLocation)
undock(s: Ship, loc: DockingLocation)
navigatewithnocargo(s: Ship, from: Location, to: DockingLocation)
Basin
Location
belongsTo : Basin
at
0..1
at
0..*
WaitingArea
DockingLocation
availablespots : Int
canrefuel : Boolean
refuelingrate : Int
dockingduration : Float
isAt
0..1
isAt
0..*
0..1
in
in
0..*
CargoItem
weight : Int
Port
Platform

Figure 2: Class diagram of the ship operations problem in petroleum ports and platforms.

ever, some of them deserve further explanation. The variables *higherfuelrate* and *lowerfuelrate* are the fuel consumption rates of the ship when navigating with and without cargo items, respectively. Even though fuel consumption rates are the same for every ship in this problem, we decided to store this information in each ship for extensibility: possible changes in ship performance in a more dynamic environment would require re-planning. The mutually exclusive variables *readytonavigate* and *docked* refer to the status of the ship, whether it is available for moving from one location to another or docked in any docking location (port or platform). Finally, *craneidle* signals if the ship is performing neither loading nor loading operations. It is used to avoid executing them concurrently.

Both *WaitingArea* and *DockingLocation* represent the information about which basin they belong to. Property *availablespots* in the *DockingLocation* is a numeric variable corresponding to how many ships are currently allowed to dock. If a vessel docks at a port or platform, this variable is decreased by 1; it is increased if an undock operation is performed. If an instance of *DockingLocation* can perform a refueling operation, then variable *canrefuel* is set to *true* and a refueling rate can be specified (e.g., 100 liters/hour at a platform and 200 liters/hour at a port). The different (un)docking durations at ports and platforms are specified in the *dockingduration* variable (since docking and undocking durations are the same in a given location, we use *dockingduration* to represent both). Here we also store the (un)docking duration for either location.

The *Global* class holds the information about the distance between the location points shown in Figure 1 and specified in Tables 1, 2 and 3. The total fuel used by ships while delivering the cargo items is stored in the global property *totalfuelused*, which defines the quality of the plan and which must be minimized by the planners. Even though the problem has a set of criteria to be optimized, in this model we evaluate only the total fuel used and the makespan. In addition, *loadingrate* holds the rate of loading and unloading cargo in the ports and platforms (1 ton/h).

We have identified eight main operators (action schema) performed by the ships, as listed below.

- *navigatewithnocargo*: Navigate from one location to a docking location without cargo. Lower fuel consumption is considered. The duration for this action is specified as ‘distance(from,to)/s.speed’.
- *navigatewithcargo*: Navigate from one location to a docking location with cargo (*currentload* > 0). Higher fuel consumption is considered. The duration is specified as ‘distance(from,to)/s.speed’.
- *navigate2waitingarea*: Navigate from one location to a waiting area. This operator considers the total fuel consumption necessary to get to the destination and then to a refueling location. *lowerfuelrate* is employed in this case. The duration is specified as ‘distance(from,to)/s.speed’.
- *dock*: Dock the ship in one of the available spots in the docking location (port or platform). The duration is specified as ‘loc.dockingduration’.

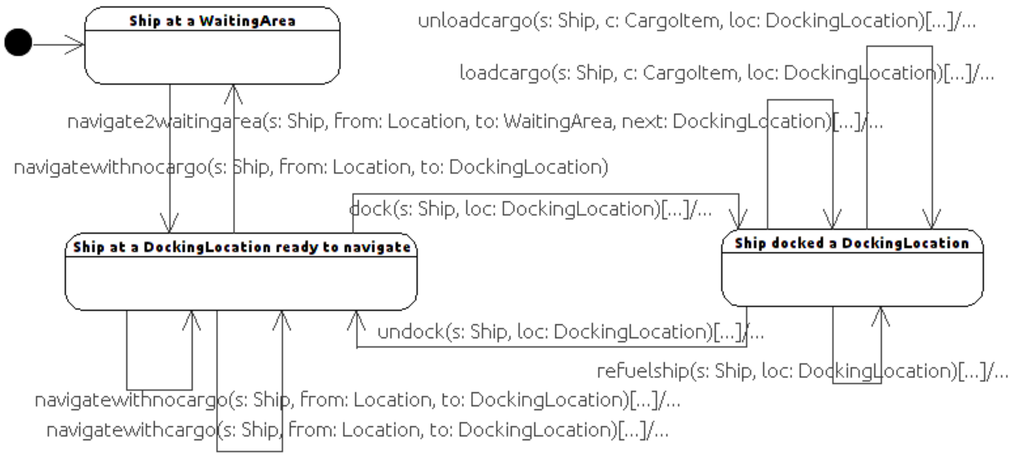

Ship at a WaitingArea
unloadcargo(s: Ship, c: CargoItem, loc: DockingLocation)[...]/...
loadcargo(s: Ship, c: CargoItem, loc: DockingLocation)[...]/...
navigate2waitingarea(s: Ship, from: Location, to: WaitingArea, next: DockingLocation)[...]/...
navigatewithnocargo(s: Ship, from: Location, to: DockingLocation)
dock(s: Ship, loc: DockingLocation)[...]/...
Ship at a DockingLocation ready to navigate
Ship docked a DockingLocation
undock(s: Ship, loc: DockingLocation)[...]/...
refuelship(s: Ship, loc: DockingLocation)[...]/...
navigatewithnocargo(s: Ship, from: Location, to: DockingLocation)[...]/...
navigatewithcargo(s: Ship, from: Location, to: DockingLocation)[...]/...

Figure 3: State machine diagram of the Ship.

- *undock*: Undock the ship from one of the spots used in the docking location (port or platform). The crane must be idle for this operation. The duration is specified as 'loc.dockingduration'.
- *loadcargo*: Load a cargo item from the location where the ship is docked. The ship must have available capacity to load the item. The crane must be idle and during the whole operation the crane becomes unavailable. The duration is specified as 'c.weight/loadingrate'.
- *unloadcargo*: Unload a cargo item from the docked ship to the location. The crane must be idle and during the whole operation the crane is unavailable. The duration is specified as 'c.weight/loadingrate'.
- *refuelship*: Refuel the ship's tank to its maximum capacity. The ship must be docked during the whole operation. The duration is specified as '(s.fuelcapacity - s.currentfuel)/loc.refuelingrate' which is the time necessary to re-fill the fuel that has been consumed.

The actions of the domain are modeled using two diagrams: the class diagram and the state machine diagram. In the class diagram, we define the name, parameters and duration for each operator (we use discrete time). The dynamics of the actions are specified in the state machine diagram, in which it is possible to represent the pre- and post-conditions of the operators declared in the class diagram. In itSIMPLE, pre- and post-conditions are defined using the formal constraint language called *Object Constraint Language* (OCL) (OMG 2003), a predefined language of UML. Usually every class in the class diagram has its own state machine diagram. A state machine diagram does not intend to specify all changes caused by an action. Instead, the diagram details only the changes that the action causes in an object of a specific class. Figures 3 and 4 show the state machine diagrams for the classes *Ship* and *Cargo*, respectively.

Timing diagrams and annotated OCL expressions are used to specify how properties change in an action horizon. For example, properties such as *readytonavigate* and *craneidle* become *false* when the action starts and then change to

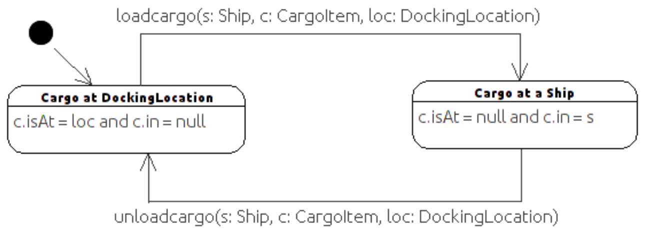

loadcargo(s: Ship, c: CargoItem, loc: DockingLocation)
Cargo at DockingLocation
c.isAt = loc and c.in = null
Cargo at a Ship
c.isAt = null and c.in = s
unloadcargo(s: Ship, c: CargoItem, loc: DockingLocation)

Figure 4: State machine diagram of the Cargo.

true when it ends. In itSIMPLE, we can represent this effect in the timing diagrams or in the OCL conditions. For example, *readytonavigate* is used to control the status of the ship when navigating from one location to another, preventing the planner from assigning another navigation action during the operation. As an effect of action *navigatewithnocargo* for instance, *readytonavigate* must be set to false at the start and then set to true at the end (when the ship arrives at the destination), as done in PDDL.

If a timing diagram is not used, temporal operators are annotated to the OCL pre- and post-conditions. For example, the variable *availablespots* is decreased as soon as the action *dock* is started, preventing any other ship docking at the same spot. This is done by annotating the post-condition 'loc.availablespots = loc.availablespots - 1' to the interval [start,start]. Users can also specify numeric intervals; however, PDDL does not support such indexation of time points. Property *readytonavigate* is also set to false when *dock* starts, 's.readytonavigate = false' in the interval [start,end]. Undocking is similar, but the variable *availablespots* is increased at the end and *readytonavigate* is set to true at the end. Therefore, we can guarantee that navigation will not be assigned during the whole process of docking and undocking. Moreover, the refueling operation must guarantee that the ship remains docked for the entire duration of the action. That is done by annotating the precondition 's.docked = true' with the interval [start,end]. In PDDL, this precondition would be be translated to '(over all (docked ?s))'.

In order to illustrate the resulting specification of the actions and facilitate their understanding, we present below the PDDL code for the actions *navigate2waitingarea* and *loadcargo*. This code was generated automatically by itSIMPLE.

```
(:durative-action navigate2waitingarea
 :parameters(?s - Ship ?from - Location ?to - WaitingArea
    ?next - DockingLocation)
 :duration
    (= ?duration (/ (distance ?from ?to) (speed ?s)))
 :condition
    (and (at start (at ?s ?from))
      (at start (readytonavigate ?s))
      (at start (canrefuel ?next))
      (at start (>= (currentfuel ?s)
        (+ (* (distance ?from ?to) (lowerfuelrate ?s))
        (* (distance ?to ?next) (lowerfuelrate ?s))))))
 :effect
    (and (at end (at ?s ?to))
      (at end (decrease (currentfuel ?s)
        (* (distance ?from ?to) (lowerfuelrate ?s))))
      (at end (increase (totalfuelused)
        (* (distance ?from ?to) (lowerfuelrate ?s))))
      (at end (not (at ?s ?from)))
      (at start (not (readytonavigate ?s)))
      (at end (readytonavigate ?s))))

(:durative-action loadcargo
 :parameters (?s - Ship ?c - CargoItem
    ?loc - DockingLocation)
 :duration (= ?duration (/ (weight ?c) (loadingrate)))
 :condition
    (and (at start (at ?s ?loc))
      (at start (docked ?s))
      (at start (>= (loadcapacity ?s)
        (+ (currentload ?s) (weight ?c))))
      (at start (isAt ?c ?loc))
      (at start (craneidle ?s)))
 :effect
    (and
      (at end (increase (currentload ?s) (weight ?c)))
      (at end (in ?c ?s))
      (at end (not (isAt ?c ?loc)))
      (at start (not (craneidle ?s)))
      (at end (craneidle ?s))))
```

In itSIMPLE, UML object diagrams are used to describe the initial state and the goal state of a planning problem instance. The object diagram represents a picture of the system in a specific state. It can also be seen as an instantiation of the domain structure defined in the class diagram. This instantiation defines four main aspects: the number and type of objects in the problem; the values of the attributes of each object; and the relationships between the objects. In our problem, the initial state consists of a set of ships at their corresponding waiting areas and with the corresponding property values, the cargo items and their respective initial locations (ports), the platforms with their available spots and refueling capability, as well as all the distances between the existing location objects (this information can be inserted by importing data in a text file as opposed to manually inputting the information). The goal state is an object diagram in which all cargo items are at their destination and the ships are back to their respective waiting areas.

Besides the object diagrams for defining initial and goal states, we also model the objective function to be optimized in every planning situation. In itSIMPLE, we select the domain variable to be minimized in a way that allows it to be represented as a linear function in the *:metric* section of PDDL. In this model we consider (1) the total fuel used (stored in the variable *totalfuelused*) and (2) the makespan. The cost of docking time of each ship is not considered in this work due to limitation on available general planners in dealing with continuous properties/time. The continuous approach could be used to compute the time that a ship remains docked for its operations, providing the necessary costs to be considered during planning.

Model Testing with Planners and Plan Analysis

itSIMPLE can automatically generate a PDDL model from a UML representation. In addition to the automated translation process, the tool can communicate with several planners in order to test the domain models in an integrated design environment. In this application, the planners must be selected based on the resulting PDDL model requirements that extend beyond the classical approaches.

In order to analyze the generated plans, itSIMPLE provides two main support tools for plan analysis: simulation and validation. Plan simulation is performed by observing a sequence of snapshots (UML object diagrams), state by state, generated by applying the plan from the initial state to the goal state. The tool highlights every change in each state transition as described by Vaquero et al. (2007). For the plan analysis, itSIMPLE provides charts that represent the evolution of selected variables such as those related to the quality of a plan (metrics). In addition, itSIMPLE provides the use of the tool VAL[1] to validate the plans generated by PDDL-based planners.

Experimental Results

We present two case studies in this section to demonstrate how planners solve the ship operations problem (in one scenario for expansion of the platforms network) using the model generated by itSIMPLE. In the first case study, we investigate the performance of three classical, modern planners using a reduced version of the model in which no time constraints are considered. We focus on plan feasibility and the minimization of fuel consumption. Time constraints usually add more difficulty to the AI P&S techniques so we aim to set up a baseline performance with such a first study. In the second case study, we analyze the output of three modern planners using the model described in the paper, i.e., with the time constraints and requirements; however, only the makespan is considered in the minimization function. In this latter study, we selected three planners that were able to read and correctly handle the PDDL durative-actions present in the model.

[1]Available at http://planning.cis.strath.ac.uk/VAL/

In both case studies, we investigate different delivery request scenarios. We analyze the performance of the selected planners in problem instances with the number of cargo items equal to: 5, 7, 9, 11, 13, and 15 (with different weights). In these instances, *P1* has n cargo items while *P2* has $n + 1$ to simulate unbalanced requests. The problem instance with 15 cargo items represents a realistic demand from the platforms. In all instances, there are six ships in *A1* and four in *A2* in the initial state–all of them with 600 liters of fuel capacity, 400 liters of fuel, 100 tons of load capacity, no cargo, an average speed of 70 km/h, 0.3 l/km and 0.2 l/km as the higher and lower fuel consumption rates, respectively. In addition to the ports, platforms *F5* and *G3* are able to perform refueling. 100 l/h is the refueling rate at the ports and at platforms *F5* and *G3*. Docking and undocking durations are set to 1 hour in the ports and 0.5 hour in the platforms.

In our experiment, planners were run on an Intel Core i7 950 3.07 GHz computer with 4.00 Gigabytes of RAM.

Case Study 1: No Time Constraints

In this case study we consider a simplified model with no time constraints. Taking into account the model in Figure 2, we do not include the variables related to time and rates such as *loadingrate*, *speed*, *refuelingrate* and *dockingduration*. Actions are adapted accordingly. In fact, they are used only in the definition of action durations.

We selected the planners Metric-FF (Hoffmann 2003), SGPlan6 (Hsu and Wah 2008) and MIPS-xxl 2008 (Edelkamp and Jabbar 2008) for this experiment. Other planners such as LPG, LPG-td, LPRPG were also tried for this experiment but they could not handle the model (e.g., the planner halts with a segmentation fault). We investigate the performance of Metric-FF and MIPS-xxl 2008 with and without the optimization flag on. To analyze the planners' performance we look at the generated plans from the six problem instances (*p05, p07, p09, p11, p13, p15*) and measure the runtime, number of actions in the plan and the total fuel used by ships. We assigned a 6-hour timeout for the planners. Table 4 shows the results from this case study.

As shown in Table 4, Metric-FF without optimization is able to provide a solution to every problem instance. However, the planner is unable to solve any problems with the optimization flag on[2] – the time limit is reached in every case. SGPlan6 is not able to solve problems *p09* and *p15*: the planner stopped before reaching the time limit. Nevertheless, SGPlan6 outperforms Metric-FF in *p07* and *p11*in terms of the number of actions and the total fuel used. Metric-FF outperforms SGPlan6 in most of the cases. MIPS-xxl 2008 in terms of the number in all problem instances.

Analyzing the plans generated by Metric-FF without optimization, we detected that even though several vessels are available for the operations, the planners provide solutions in which just a few ships are used. For example, in the plan generated for the problem *p05*, only one ship (S9) is used for all deliveries and transportations. Only three ships (S7,S8,S9) are used to solve the problem *p15*. In addition, some plans contained unnecessary consumption of fuel, for example in cases where a ship travels from location A to B and then from B to C without doing any delivery, while it could go directly from A to C using less fuel (shorter distance). SGPlan6 shows a similar behavior by using a few ships to solve the problems; however, it does not show the unnecessary fuel consumption behavior.

[2]Since Metric-FF is treated as a blackbox in this experiment, we did not explore the reasons for why it does not solve any of the problems.

Case Study 2: With Time Constraints

In this case study we consider the complete model of ship operations in the port and platforms, with time constraints and requirements, illustrated in Figure 2. We selected planners POPF (Coles et al. 2010), SGPlan6 and MIPS-xxl 2008 for this experiment. POPF participated in the seventh International Planning Competition (2011) in the deterministic, temporal satisficing track. We have set up POPF to generate as many solutions as it could in the time limit, improving the plan quality (makespan in this case) in each subsequent solution. Other planners such as LPG-td and LPRPG were also tried for this experiment but they could not handle the model. To analyze the selected planners' performance we looked at the generated plans from the six problems instances (*p05, p07, p09, p11, p13, p15*) and measured the runtime, number of actions in the plan, the total fuel used by ships and the makespan. We assigned a 3-hour timeout for the planners to simulate a more realistic response horizon. Table 5 shows the results for the second case study.

As shown in Table 5, SGPlan6 is the only planner in this experiment that managed to solve some of the instances. Surprisingly, the more recent planner POPF does not solve any of the problem instances. We have also checked smaller problems with 2 and 3 cargo items, and even 1 cargo item and 1 ship, but it still does not solve them. SGPlan6 produced exactly the same solutions as in case study 1; the runtimes were greater in most of the cases though.

Discussion

The case studies showed that an AI P&S approach for solving the ship operation problem in Petrobras is possible; however, the available domain-independent planners do not currently provide the necessary set of tools to solve the modeled problem in real life. SGPlan6 can often provide a feasible solution, but optimal solutions in a realistic horizon do not appear to be achievable. As opposed to modeling the problem using optimization approaches (e.g., using MIP or CP models), our intention was to develop a model in order to evaluate if current planners would have acceptable performance in real scenarios. From the results presented in the previous section, we conclude that the planners do not succeed at this task.

Since one of the main goals in this paper is to describe the modeling experience, in this investigation we tried to model the problem using KE tools that would direct the model to standard representation languages in AI P&S and therefore could potentially be read by several planners. In fact, modeling the problem in UML and then translating to PDDL was

Cargo	Metric-FF						SGPlan6			MIPS-xxl 2008		
	no optimization			with optimization			With Metric			With and without optimization		
	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)
5	13.56	42	631	Timeout	-	-	12.11	47	781	Timeout	-	-
7	38.50	59	881	Timeout	-	-	933.68	58	805	Timeout	-	-
9	106.94	76	1,073	Timeout	-	-	X	-	-	Timeout	-	-
11	244.45	88	1,523	Timeout	-	-	1,114.42	96	1457	Timeout	-	-
13	284.68	105	1,533	Timeout	-	-	1,041.22	108	1630	Timeout	-	-
15	499.47	122	1,844	Timeout	-	-	X	-	-	Timeout	-	-

Table 4: Results from Case Study 1 - No Time Constraints. 'Timeout' means that the planner reached the 6-hour limit without generating any plan. 'X' means that the planner stopped before reaching the timeout limit without generating a plan.

Cargo	POPF				SGPlan6				MIPS-xxl 2008			
	Runtime (s)	# actions	Fuel (l)	Makespan (h)	Runtime (s)	# actions	Fuel (l)	Makespan (h)	Runtime (s)	# actions	Fuel (l)	Makespan (h)
5	X	-	-	-	1.82	47	781	152.52	Timeout	-	-	-
7	X	-	-	-	3,071.61	58	805	294.34	Timeout	-	-	-
9	X	-	-	-	X	-	-	-	Timeout	-	-	-
11	X	-	-	-	3,245.85	96	1,457	422.24	Timeout	-	-	-
13	X	-	-	-	1,180.81	108	1,630	600.54	Timeout	-	-	-
15	X	-	-	-	X	-	-	-	Timeout	-	-	-

Table 5: Results from Case Study 2 - With Time Constraints. 'Timeout' means that the planner reached the 3-hour limit without generating any plan. 'X' means that the planner stopped before reaching the timeout limit without generating a plan.

feasible and practical. The semantics of the model results in a natural mapping between real objects and objects in the model. Moreover, the mapping of the generated solution follows the same rules and has a direct map to the real world. This modeling ease is not necessarily true in the models developed with optimization technology.

It is indeed possible to refine and adapt the model so that planners could run faster and produce better solutions. A designer could even reduce the problem to a basic form so other planners can handle it. However, we tried to perform the modeling process by focusing on the semantics of the model – keeping the mapping obvious for non-planning experts. In fact, the resulting model can be seen as a transportation problem (the class of problems addressed the most by the AI Planning community) with extensions that make it more realistic (e.g., load capacity, fuel capacity). The model does not seem to be far different from what we see in classical numeric and temporal domains (e.g., logistics, depots, driverlog, zenotravel, etc.), but it indeed combines certain requirements that test the limits of the state-of-the-art planners. Therefore, it is a challenge domain for AI P&S approach. That is why it has been proposed as one of the challenge domains in the ICKEPS'12 competition.

Conclusion

In this paper, we have investigated a real planning problem, the planning and scheduling of ship operations in ports and platforms, using an AI P&S approach. We described the design process used for building a domain model with the KE tool itSIMPLE. In order to validate the model and investigate the applicability of state-of-the-art planners in this problem, two case studies were conducted. The first one considers a semi-realistic scenario in which no time constraints are considered and the second brings a more realistic case in which time is considered. The planners were selected based on their capacity in dealing with the domain model requirements (durative-actions, numeric variables, and metrics). The metrics considered in these problems focus on the minimization of different parameters such as total fuel used by ships and the makespan.

Experimental results showed that in both cases some planners can provide valid solutions for the problem, however, they struggle to provide solutions to more realistic problems. It is important to note that few planners can deal with such a combination of PDDL features. Therefore, the resulting PDDL model brings interesting challenges even for the state-of-the-art planners. The model will be made available in order to share our results on this domain. In addition, experience from this application has motivated the improvement of itSIMPLE towards time-based models to support designers on real-world problems.

Acknowledgment

The first author is supported by the Government of Canada Post-Doctoral Research Fellowship. The second author is supported by FAPEAM.

References

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Edelkamp, S., and Jabbar, S. 2008. MIPS-XXL: Featuring External Shortest Path Search for Sequential Optimal Plans and External Branch-And-Bound for Optimal Net Benefit. In *Short paper for the International Planning Competition 2008*.

Hoffmann, J. 2003. The metric-FF planning system: Translating ignoring delete lists to numerical state variables. *Journal of Artificial Intelligence Research (JAIR)* 20.

Hsu, C.-W., and Wah, B. W. 2008. The sgplan planning system in ipc-6. In *Proceedings of the Sixth Internation Planning Competition (IPC) in ICAPS 2008*.

Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *In Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 526–533. AAAI Press.

Kautz, H., and Walser, J. P. 2000. Integer optimization models of ai planning problems. *The Knowledge Engineering Review* 15:2000.

OMG. 2003. *UML 2.0 OCL Specification m Version 2.0*.

OMG. 2005. *OMG Unified Modeling Language Specification, m Version 2.0*.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated tool for designing planning environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA*.

Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From requirements and analysis to PDDL in itSIMPLE3.0. In *Proceedings of the Third ICKEPS, ICAPS 2009, Thessaloniki, Greece*.

Vaquero, T. S.; Silva, J. R.; and Beck, J. C. 2011. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the ICAPS 2011 workshop on Knowledge Engineering for Planning and Scheduling workshop. Toronto, Canada*.