Solving Logistic Problem with Multi-Agent System

Conducted in collabration with the Odense Steel Shipyard Group

Ali Cevirici
cevirici@mip.sdu.dk

Henrik Møller-Madsen
henrikmm@mip.sdu.dk

Master Thesis in Computer Systems Engineering
The Maersk Mc-Kinney Moeller Institute - MIP
University of Southern Denmark - SDU

Supervisor: Kasper Hallenborg
External supervisor: Niels J. Jakobsen

May 11, 2007

Abstract

Danish as well as international corporations increasingly notice that they are operating in dynamical environments, wherein high demands to flexibility and reorganization capabilities are set. Demands, which are initialized partly coursed by an increasing competition, resulting from a still faster evolution of technology and varying demands from customers as well as the internal distribution of work.

The Odense Steel Shipyard (OSS) is one of the companies facing this tendency by their logistic problem, concerning transportation of large ship blocks locally at the shipyard. OSS is taking part in a consortium, with other leading companies and research institutes in Denmark; The DECIDE project. The objective of DECIDE is to investigate the Multi-Agent technology and emulation.

Over the past decades the abstraction level of software engineering have progressed in order to model complex and large-scale problem domains, wherein human actors today play an important role.

This thesis explores how to adapt multi-agent technology to solve the logistic planning at OSS. Two Multi-Agent Systems(MAS) have been designed in the DECAF and Cougaar frameworks respectively, and a 3D simulation model of the OSS domain was developed.

Acknowledgements

We will like to thank all the people at OSS that have helped us in this thesis project. Special thanks to Henning Klitten Jensen, who is a B-planner at OSS, for providing the data we needed in order to design the simulation model, also a special thanks to our external supervisor Niels J. Jakobsen for his assistance at OSS. Furthermore we would like to thank Claus Rønaa(D-planner) for letting us whatch him in action, and Ole T. Sørensen(C-planner), for letting us attend to some of his morning meetings at OSS. We will also like to thank everyone in the DECIDE project, who has contributed to our project with input and disscussion at the DECIDE seminars, especially SimCon for giving us basic insight in AutoMod.

Last, but not least, we would like to send a special thanks to our supervisor Kasper Hallenborg, for his guidance, advice and patience in this thesis project.

Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

DECIDE Project

The DECIDE project is a consortium which focuses to establish a development co-operation between the partners of the consortium, regarding Multi-Agent Systems(MAS) and Emulation as paradigms for a more robust optimization of production and logistic [4]. The purpose is to investigate multi-agent systems an emulation in a theoretical level to show that these technologies, compared to traditional planning and control systems, creates more robust and optimal plans. This investigation should be visualized to the people in the production environment, from decision-makers to daily users.

1.1 Partners

In this section, we will give a brief description of the partners in the DECIDE project.

- **RoboCluster** is a growth initiative for the robotics and automation industry in Southern Denmark[1].
- **FKI logistex** is one of the companies that are leading in the area of high capacity baggage handling systems for airports. It is one of the few companies which can deliver complete solutions to large airports.
- **Bang & Olufsen** produces audio- and video equipments which are known for:
  - Outstanding Performance
  - Durability
  - Classic Design
  - Long-term Reliability

[1]http://www.robocluster.com/english/index_html

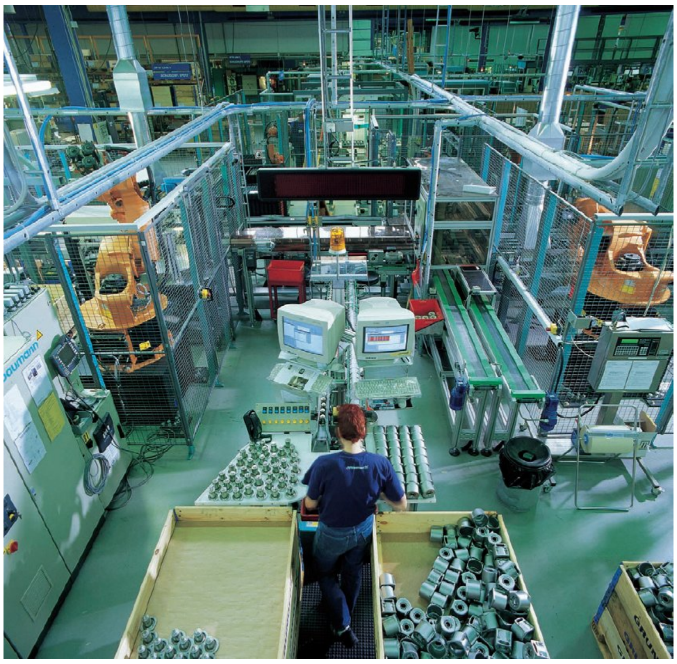


Figure 1.1: Grundfoss

The aluminium parts that are used in the surface of the products has true Bang & Olufsen characteristics .The machining, polishing and anodising of aluminium have been a strategic competence within the company for many years [5].

- **LEGO** has it's head office in Billund, Denmark. The name 'LEGO' is an abbreviation of the two Danish words "leg godt", meaning "play well"[2]. The company produces LEGO toy bricks for children, which are sold in more than 130 countries.
- **Odense Steel Shipyard**(OSS) is one of the worlds most modern shipyards, which is able to build any kind of ship to the international shipping. OSS has through the last couple of years mainly build large container ships. OSS is a part of the OSS Group which contains following four shipyards
  - Odense Steel Shipyard Ltd
  - Volkswerft Stralsund GmbH
  - Baltija Shipbuilding Yard JSC
  - Loksa Shipyard Ltd.

[2]http://www.lego.com/eng/info/default.asp?page=group

and one machine factory.

  - Balti ES Ltd.

- **MESH-Technologies** is a software company, that works within the are of High Performance Computing and Distributed Systems. The company produces software to classical supercomputers and loosely cupled systems, including GRID systems and Pervasive Computing.
- **The Maersk Mc-Kinney Moller Institute for Production Technology**(MIP) has the aim to become a highly technological, internationally recognized centre of excellence, where academia and industry in close collaboration develop new technologies for intelligent autonomous systems[3].
- **IMADA** has kompetences within effective algorithms and high performance computing in the area of computer science.
- **The technical faculty at SDU**[4] has kompetences within hardware and software development to embedded systems.
- **Simcon** is a software and consultancy bussiness[5] which offers complete knowledge based solutions within the main fields:
  - System emulation
  - Concept evaluation
  - Simulation and Animation
  - Optimization
  - Software development
  - Training
- **Danish Technological Institute** occupies a crucial position at the point where research, business, and the community converge. The Institute's mission is to promote growth by improving interaction and encourage synergy between these three areas[6].
- **Fyn Enterprise Development Centre**[7] is the host organization for EU Center Fyn, which has kompetences within project description, identification of EU-support and funding possibilities, and EU-project applications.

[3]http://www.mip.sdu.dk/general_information/
[4]formerly known as IOT
[5]http://www.simcon.dk
[6]http://www.danishtechnology.dk/6158
[7]Fyns ErhversCenter

- **Grundfos** is one of the world's leading pump manufacturers. The mission of the company is to develop, produce and sell high quality pumps and pumping systems world-wide.[8]

1.2 Cases

Some partners of the consortium has specific cases that will be investigated during the DECIDE project. These cases are described briefly below.

- **FKI logistex**
  This case concerns Baggage Handling Systems(BHS) for airports. BHS at airports are generally used for baggage that are received from arriving planes and baggage that are delivered long before departure by passengers[29]. The baggages in a BHS are transported by totes in a conveyor system or by Destination Coded Vehicles (DVC). An example setup of a conveyor system for a BHS can be seen in figure 1.2.

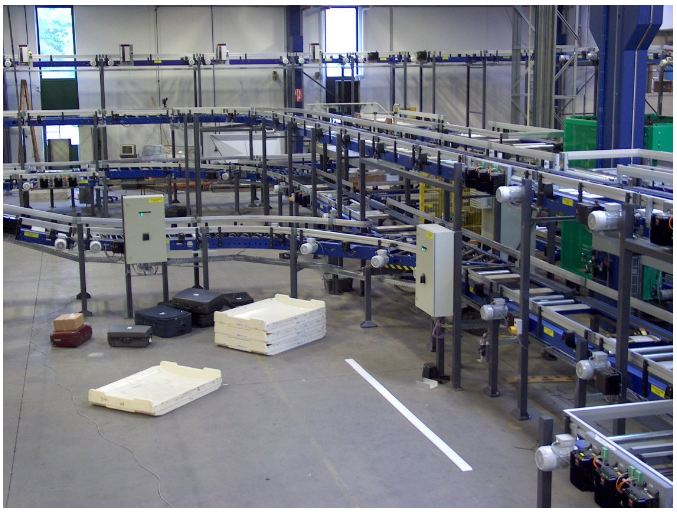


Figure 1.2: BHS hardware setup example

  Many problems can arise in a BHS, which is a very dynamic environment, such as delayed arrivals, missing tag codes, flight changes, break-downs and peak loads.

- **Bang & Olufsen**
  This case concerns the anodizing plant for surface treatment of aluminium at B&O. The anodizing plant consist of a bufferline with 55

[8]http://www.grundfos.com/web/grfosweb.nsf

baths where elements hang up on element-bars are dipped in. Today the throughput are 7.5 element-bars per hour. The goal is to reach a throughput of 8.0 - 8.5 element-bars per hour[9]. A part of the anodizing plant is shown in figure 1.3.

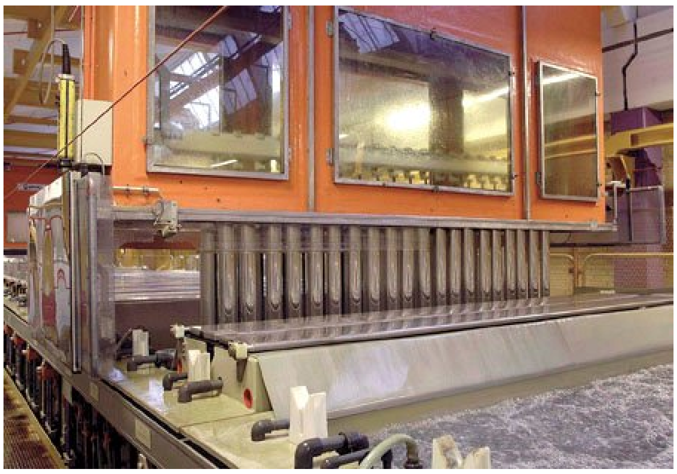


Figure 1.3: Part of the anodizing plant at B&O

- **LEGO**
  This case concerns separating a pile of plastic bags containing LEGO bricks, placed on a conveyer belt as shown in figure 1.4.

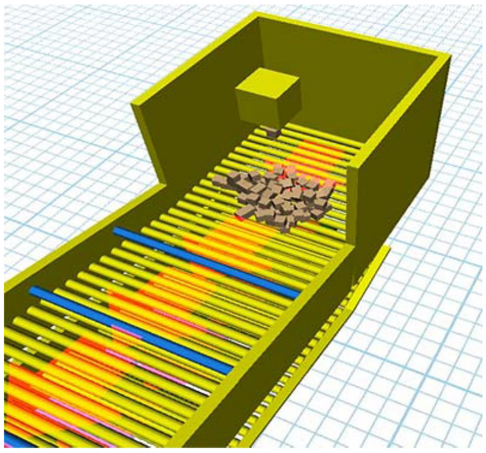


Figure 1.4: Conveyor belt at LEGO, simulated with SimCon Experior

[9]Powerpoint slide with title: Bang og Olufsen   Anodiseringsanlæg

- **Grundfos**
  This case concerns the process of kataforese coating pumps. The metal[10] pumps are kataforese coated, by being placed in different kind of baths with chemical liquids. Two cranes in the production plant moves bars with pumps and kataforese coates them. The movement of the cranes are synchronized to avoid collission, but this makes the production plant ineffective, because cases arise where only one crane is coating pumps and the other crane goes through a kataforese process without any pumps at all.

- **Odense Steel Shipyard**
  This case concerns the coordination of the daily ship block transportations at Odense Steel Shipyard(OSS). Figure 1.5 shows a KAMAG vehicle, transporting a ship block. This case is investigated by us with this thesis.

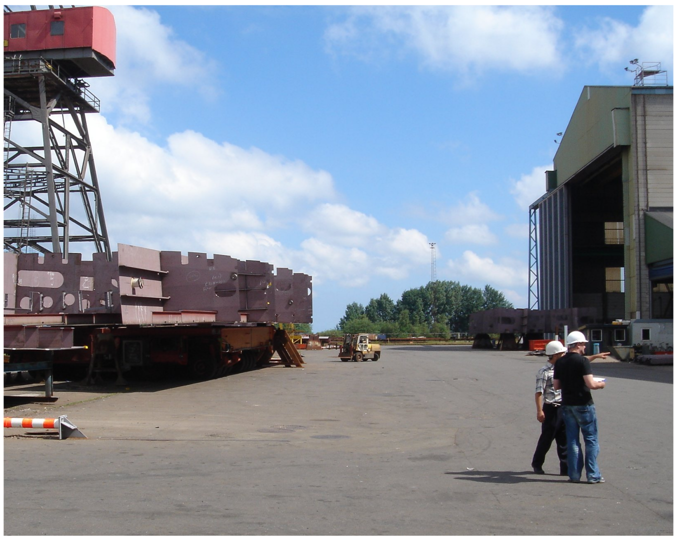


Figure 1.5: A KAMAG vehicle transporting a ship block

[10]either aluminium or iron

Chapter 2

Project Description

This chapter describes the thesis case, and the investigation of the OSS problem domain. Furthermore we describe the overall system requirements, and the project demarcation, wherein we state our objectives with this thesis project. We finalize this chapter with a description of the project workflow.

2.1 Case study

Danish and international companies see an increasing tendency towards operation in dynamical environments, wherein great demands to flexibility, reorganization and adaptation takes place. Requirements which are initialized partly from an increasing competition on the international marked, with in particular as consequence of a still faster evolution of technology and varying demands from customers as well as internal departments of firms.

The Odense Steel Shipyard Group(**OSS**) experiences the following logistic problem in the daily scheduled work.

The logistic problem concerns transportation of elements between different internal locations on OSS. Some elements are produced by OSS while other elements are being produced externally. All elements go through a process in the environment at different locations. When assembled all the elements form a whole ship.

Transportation of elements are conducted using heavy specialized machinery vehicles called KAMAG vehicles. These specialized vehicles have a payload capacity of approximately 500 ton each, which makes them suitable for the given task at OSS.

The transportation itself is conducted through wide roads, which connects the different locations, where elements are to be processed or stored away for future use.

Some road width's are limited (figure 2.1 shows the road system), resulting in complicated passage of KAMAG vehicles carrying heavy loads, thus coordination of the vehicles are required.

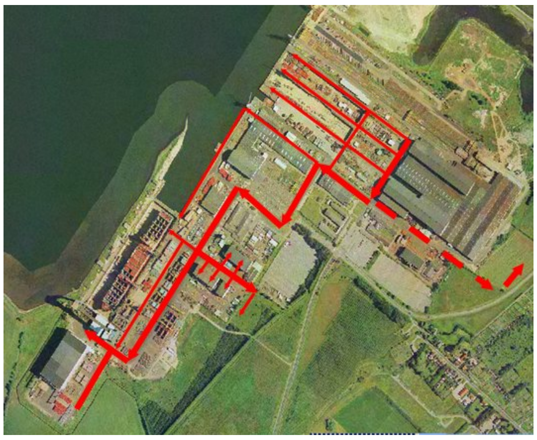


Figure 2.1: Aerial overview of the road system at OSS

Organization of the transportation is similar to a cab-system. All supervisors also referred to as C-planners, which all lead a well defined area on OSS, call in every day at approximately 9 o´clock reporting which transportation they need that day. This information is then processed by the D-planners(usually only one), resources (KAMAG vehicles) are allocated and coordinated in a logistic plan for that day. The logistic plan is a grand schedule plan of all transportations, which include information regarding what ship blocks to collect, where and when to collect them, and where to place them. The plan is then reported to the supervisors, which gives them a reference schedule for the rest of the day. Although an overall plan is formed decentralized changes to the plan are considered inevitable.

Along the day different scenarios will affect the scheduled plan, where the main scenarios are listed below:

- **Production deviation** such as delays in production, or if elements are done prior schedule, thus needing transportation earlier.

- **Transport bottlenecks** occurs if two or more KAMAG vehicles must wait on each other. Planning is done to prevent this type of situation, but in real life it is practical impossible. Other times a KAMAG vehicle is forced to pass narrow paths with the greatest precision, increasing transportation time.

- **Breakdowns** result in an unaccessible KAMAG while it is being repaired. The KAMAG is then transported to a special place where it is repaired. The heavy ship blocks are very hard on KAMAG vehicles, which is why they breakdown at times.
- **Misplacement** of other kinds of transportation vehicles also occurs, which block the traffic of KAMAG vehicles. The solution here is to find the driver of the vehicle, who is blocking the way and remove the vehicle. Other times, storage locations are occupied with unreported scaffolds, meaning that a new storage location must be found for the ship block, thus replanning is demanded.

Unpredictable events are coordinated with the corresponding C-planners responsible for the affected areas. Currently it is a problem that consequences of choices are unknown.

2.2 Researching and understanding the case

This section is the result from a data collecting research phase we have conducted at OSS over several months. The current section will serve to give an overview of the areas on OSS which play an important role in respect to the current transportation system. In order to make a solution proposal based on multi-agent technology we will categorize the collected data and observations from OSS. This section will serve as a preface to the overall analysis.

2.2.1 Planning and logistics

Building the world's largest container ships concerns production and mounting of approximately 150.000 steel elements, 30.000 larger components and 11.000 pipes. All together a surface of 390.000 m^2 must be painted, 700km of lines must be welded and 230km of electronic wiring must be mounted. All this requires planning and logistics[1].

Due to the above requirements, OSS has developed their own computer based planning system called DPS, which is used by planners at OSS to plan various activities, such as handling manpower as well as area disposal.The program **P**roduction **M**anagement **S**ystem (**PMS**), developed internally by OSS, is used to manage steel production. In the equipment do-

[1]http://www.oss.dk

main the system ERP BaaN is used.

The scheduling at OSS is hierarchically structured as seen in figure 2.2.

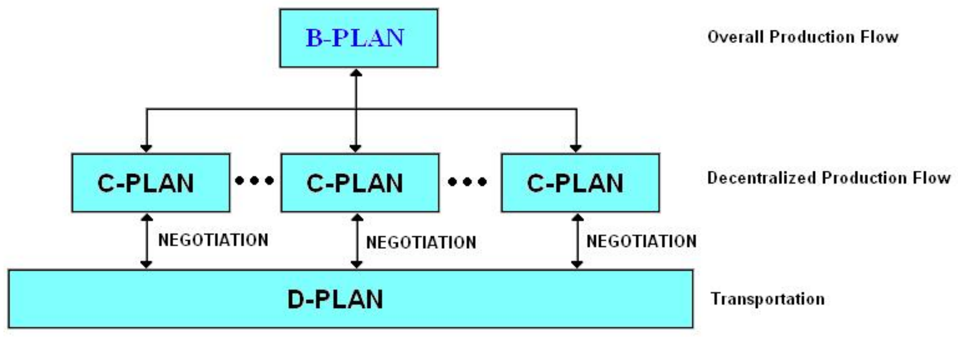


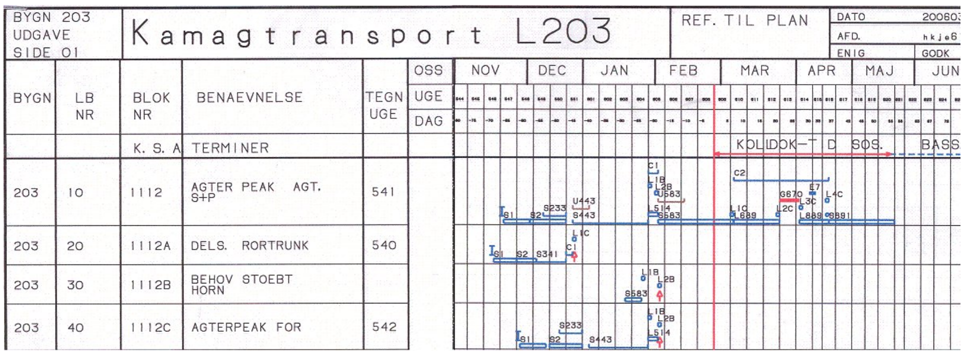
Figure 2.2: Hierarchical scheduling structure

A grand B-plan has the overall view of ship block flow in the system. The C-plans contain area specific details regarding inflow and outflow of a given area. The D-plan is a coordination of transportations from C-plans. In the following sections we will describe processes in the B-plan, C-plan and D-plan.

2.2.2 The B-plan

B-planners have got the breadth of view over the transportation on OSS. They have an idea of which blocks will be transported on which day, but not at which time. It is the B-planners, who make the B-plan, which contains an overview of all the ship blocks i.e which blocks are done, when and where. One can according to the B-plan follow any ship block through the entire ship building production flow. The process can be compared to assembly or production lines, where all the building blocks are assembled at certain assembly points(halls and storage locations), and in the end form a complete ship in the dock.

Our primary contact on OSS regarding the B-plan has been Change Agent Henning Klitten Jensen. Henning uses a program called DPS, which is used by B-planners, to design the B-plan. An extract from the B-plan is showed on figure 2.3.



BYGN	LB NR	BLOK NR	BENAEVNELSE	TEGN UGE
		K. S. A	TERMINER	
203	10	1112	AGTER PEAK AGT. S+P	541
203	20	1112A	DELS. RORTRUNK	540
203	30	1112B	BEHOV STOEBT HORN	
203	40	1112C	AGTERPEAK FOR	542

Figure 2.3: Extract of B-plan

We will not go into a detailed description of figure 2.3, but merely note that the B-plan describes the flow of any ship block from supplier to the completed ship. The B-plan implicit gives a survey of the main transportation of ship blocks internally at OSS. The C-plans are designed from the foundation formed by the B-plan. In contrast to the B-plan, which provides the overall view, the C-plans individually covers limited well defined areas on OSS.

2.2.3 The C-plan

Every C-planner has the responsibility for a specific area at OSS and his purpose is to coordinate and inspect this area, and to order vehicles to transport ship blocks to and from his area. In order to secure a smooth inflow and outflow of ship blocks, C-planners make their own C-plans ½ a year forward and modifies it as deviations from the scheduled plan occur. The C-plans are based on the B-plan and adjusted locally by C-planners to be more realistic. The C-planner inform the Production managers[2], if the C-plan deviates too much from the B-plan.

The C-planners depend on each other's deadlines, furthermore do C-planners prioritize transports, depending on where the ship blocks are going to be transported to/from; e.g. transport of ship blocks to the gantry crane has higher priority than transport of ship blocks from painting halls to storage locations.

The C-planner rides his bicycle to check the storage locations, that he is responsible for, to see which ship blocks are placed where, before he makes his day-report. The C-planner then holds a meeting with the foremen to

[2]B-planners

follow-up how far they are from finishing the ship blocks. The meeting are based on the day-report, and every location in the area, that the C-planner is responsible for, is gone through one by one. An example of such a report with placement of ship blocks can be seen on figure C.1 in appendix C.

At the end of the meeting the C-planner contacts the D-planner and orders transportations of ship blocks for that day. The C-planner is however still able to cancel/change the order within the day, if the ship block is finished before/after schedule.

2.2.4 The D-plan

The responsibility of the D-planners are to coordinate all KAMAG transportations at OSS. A typical sequence of requests from C-planners during one day is seen below and in figure 2.4:

- B6 by phone
- Hal Øst by phone (multiple changes on daily basis)
- Painting hall by e-mail
- B9 by e-mail
- B4 by phone (multiple changes on daily basis)
- Hal Syd by phone

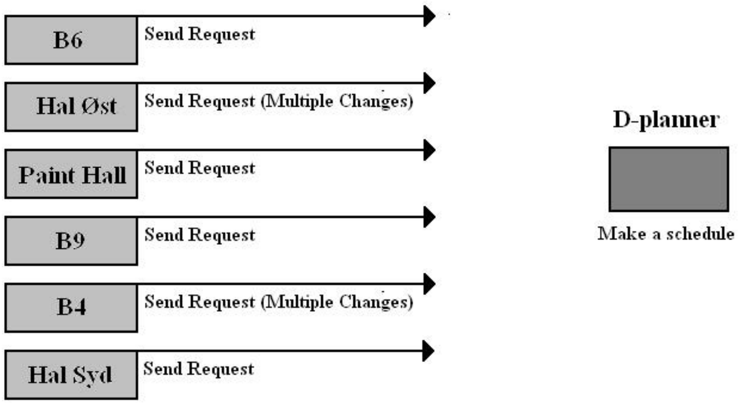


Figure 2.4: C-planners negotiating with D-planner to fulfill their needs

Claus Rønaa is one of the highly skilled D-planners and must keep track of all transportations. He makes a logistic schedule, which is modified frequently on a daily basis according to deviations received from C-planners.

Type	Vehicle No.	Payload Capacity	Dimension	Speed
KAMAG	4843	530 T	9,7m x 13,5m	Average 6-7 km/h
KAMAG	4846	450 T	9,7m x 13,5m	Average 6-7 km/h
KAMAG	4847	580 T	9,7m x 13,5m	Average 6-7 km/h
Tarok	4848	365 T	6,1m x 21m	Average 6-7 km/h
Tarok	4849	550 T	6,6m x 20m	Average 6-7 km/h

Table 2.1: Specifications of the KAMAG vehicles at OSS

Description	Combination	Payload Capacity	Speed
Normal Double	**4846** + **4847**	ca. 1030 ton	ca. 5 km/h
Large Double	**4843** + **4847**	ca. 1110 ton	ca. 5km/h

Table 2.2: Combinations used to carry heavy ship blocks of ca. 1000 ton

Claus has got a lot of experience with scheduling transportations, thus a lot of his decisions are based on intuition and experience. Whenever he encounters new ship blocks, which he has not yet transported before, due to the fact that new ships are being build, he takes a look at the blueprint of the block in question and reads the specification. This information tells him how ship blocks should be placed and how they should be supported. Table 2.1 lists the various KAMAG vehicles, that are being used to transport the many ship blocks. KAMAG vehicles can be combined to increase the overall payload capacity as seen in table 2.2.

A D-planner must keep track of all transportations, to make an overall schedule, trying to fulfill the requirements of the C-planners. As mentioned earlier the plan is destined to undergo frequent changes due to the dynamical environment. Alterations in the plan given by C-planners, must also reflect changes in the overall schedule of transportation for that given day.

To navigate KAMAG vehicles under the transportation of heavy loads, trucks are being used. Trucks at OSS have the following numbers: 4783, 4789, 4793 and 4794.

In the unit hall(N4), seen in figure 2.5 ship blocks are equipped. The space in the unit hall is limited, such that a KAMAG vehicle only can access the blocks from one side of the hall, which results in an operation similarly to a stack operation. Like when you have a stack of plates, you must first remove the top plate before moving the second et cetera. As temporary storage road T111(see figure 2.5) is used, until the correct block has been removed from the unit hall, and the correct one is placed inside the unit hall, thus this operation must be done in the second or third turn usually after 17 o´clock, since the road will be blocked while this procedure is conducted. After this operation the road is once again cleared.

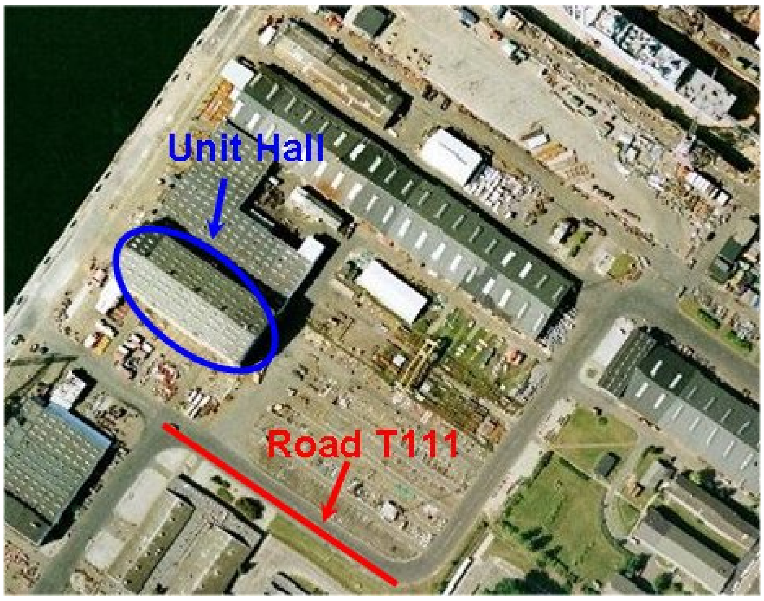


Figure 2.5: Aerial photo of the unit hall and road T111

2.2.5 Areas and Production Flow

OSS has provided a dxf file (CAD file) containing everything on OSS, such as buildings, storage locations, bushes, cranes and a lot of other information. Much of the mentioned information is of no use to this project, thus the amount of information in the file must be limited. Only buildings, storage locations and roads, where KAMAG vehicles have access are of interest to this project, thus essential information have been extracted from this file into a simplified file, containing only the following entities of interest:

- Road (this covers all roads which KAMAG vehicles can safely access)
- Paint hall (building)

- Hal Syd (building)
- Hal Øst (building)
- Unit hall (building)
- Depository (area)
- Depository with supply (area)
- Area A,B,C,D and E

The Production flow is characterized by a dynamic production course. Steel plates and profiles are cut and then assembled to sections, which are then painted, equipped before setting up to a ship in the dry dock. From this point on the ship will be completed in the equipment quay. The total building activity is approximately 10-11 months[3]. The flow of ship blocks at OSS domain runs through the production steps 1 to 11 seen in figure 2.6, for more details regarding each production step see section B.3.

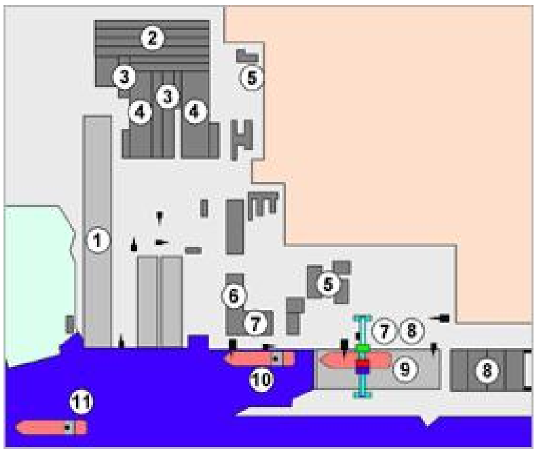


Figure 2.6: Abstract Overview of OSS

2.3 Domain Entities

In this section we describe the various physical entities from the OSS domain that are relevant to this thesis project, which includes the following:

- Transportation vehicles known as KAMAG vehicles

[3]source: www.oss.dk

- Storage locations
- Ship blocks
- Buildings (some for functional purpose others for visualization purpose only)
- The road system

2.3.1 Transportation vehicles

Transportation vehicles called KAMAG vehicles (figure 2.7), are used for transportation of ship blocks, and have the following physical properties and features:

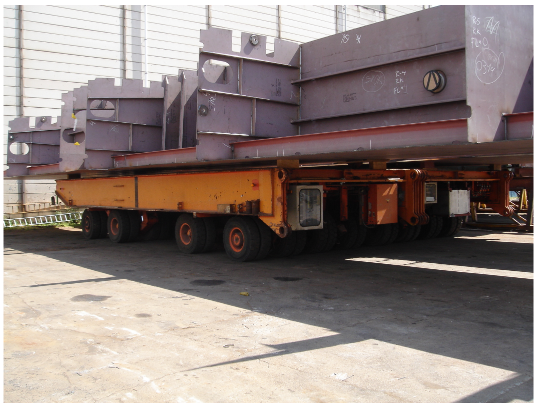


Figure 2.7: KAMAG - transporting a ship block

1. A KAMAG can only drive on the road system [4]
2. A KAMAG has a max lift capacity, limiting the weight of ship blocks to be lifted
3. A KAMAG has a maximum payload capacity, limiting the weight of ship blocks to be transported.
4. A KAMAG can transport one or multiple ship blocks
5. A KAMAG has a maximum velocity, acceleration and rotation speed
6. A KAMAG can rotate 360 degrees with any center point of rotation

[4]see figure 2.1

7. By combining two KAMAG vehicles the total lift and transportation capacity increases, thus adding the capacity of the two KAMAG vehicles
8. A KAMAG can pick up a ship block
9. A KAMAG can put down a ship block
10. A KAMAG can transport a ship block from one location to another
11. A breakdown may happen, forcing the KAMAG out of action
12. A KAMAG requires regularly maintenance, which implicitly puts the KAMAG out of action
13. A KAMAG can transport another KAMAG in case of breakdown.

2.3.2 Storage locations

Storage locations at the shipyard are used to store ship blocks, waiting for further processing at another location. Some storage locations are also used for equipping the ship blocks. Storage locations are mostly outdoors at OSS as seen in figure 2.8, but in some buildings as seen in figure 2.9, ship blocks are being build and/or equipped. There are certain properties limiting which ship blocks, that can be placed at a given storage location, those properties are listed below:

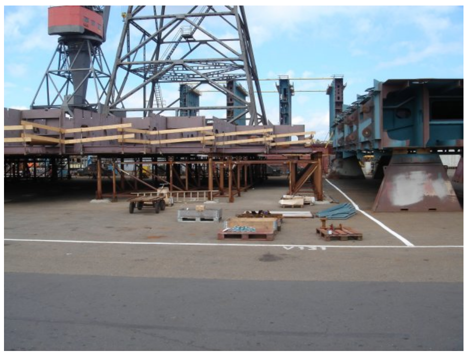


Figure 2.8: Storage location V131 at OSS

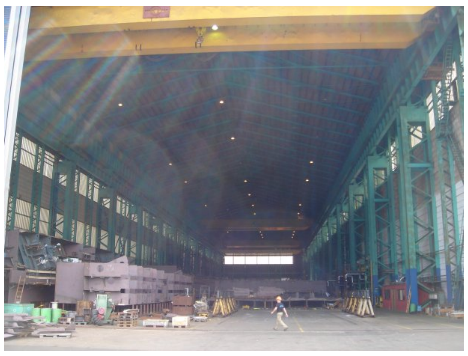


Figure 2.9: Building with storage at OSS

- A storage location has a specific geographical location.
- Width
- Depth

- Height (usually unlimited, unless placed under a roof)
- Maximum payload due to the foundation
- Some have supply, such as electricity and gas for welding

2.3.3 Ship blocks

The ship blocks like the one in figure 2.10 are the main entities at OSS in respect to the transportation problem. The ship blocks are transported from buildings and storage locations, where they are being produced to other buildings and storage locations, where they are stored, equipped and/or painted. The ship blocks progresses through a flow in the system, and are finally assembled to form a container ship at the dock as seen in figure 2.12, which is the final destination for every ship block in the OSS domain.

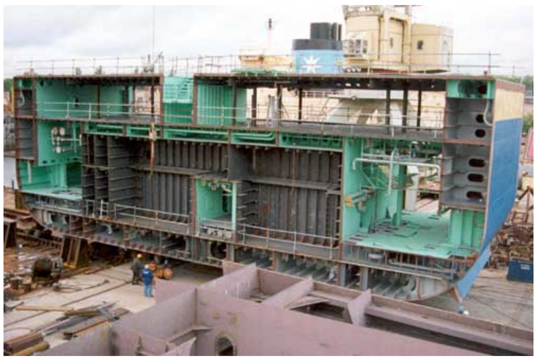


Figure 2.10: A steel section

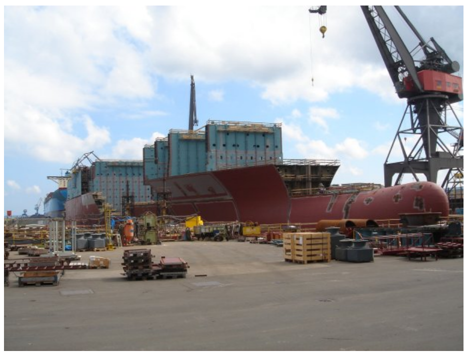


Figure 2.11: Steel sections assembled at the dock

Figure 2.12: Container ship assembled at the dock

A ship block has the following properties:

- Height
- Width
- Length
- Weight
- Type (a number indicating, which ship block it is)
- Ship blocks goes through a certain flow in the system
- A grand block is a special type of ship block that is composed from multiple ship blocks.

2.3.4 Buildings

Some buildings at OSS are of importance in respect to the transportation problem, such as halls for producing, equipping and painting ship blocks. A painting hall and a production hall are shown in figures 2.13 and 2.14 respectively.

Figure 2.13: A paint hall - painting ship blocks

Figure 2.14: The South Hall - Producing ship blocks

Following buildings from the OSS domain are of relevance:

- **Production halls:** B4, B6, B9
- **Equipment hall:** N4
- **Painting halls:** Kab1 - Kab8

2.3.5 The road system

Figure 2.1 illustrates where KAMAG vehicles can drive at OSS. Some of the roads vary in size, meaning that KAMAG vehicles may not be able to pass each other if the road is to narrow.

2.4 Overall system requirements

The system is intended to support, the daily planning of the internal transportation of ship blocks and more specifically, decision-makers, in deciding which transportation vehicle to use for transporting a certain ship block from one location to another at a given time.

This case is part of the DECIDE project as described in chapter 1, which focuses on the usability of Multi-Agent Systems to solve logistic and production related problems. We will therefore approach this case in terms of multi-agent terminology.

As input to the system, the supervisors also known as C-planners should be able to request multiple transportations on a daily basis. The system should then generate a logistic plan, also referred to as a D-plan in the OSS domain.

The system should at all times visualize the current status of domain entities, such as ship block and KAMAG vehicle placements, meaning the dynamic entities in the system. Static entities, such as storage location, buildings and the road system, should likewise be visualized at all times. Users of the system should be able to monitor the consequences of their decisions online while working. The system should be decentralized, and therefore function across a local area network.

Data consistency and robustness is of major importance, thus the system should be reliable even if parts of the system breaks down.

The user interface should be user friendly, such that it can be used by employees at OSS.

The system should be provided with information regarding the entities that reside in the OSS domain, such as the initial placement of KAMAG vehicles and the ship blocks, including their physical properties.

2.5 Demarcation of project

There are many similarities between the agent paradigm and situations described in the case. Supervisors also known as C-planners, behave like decentralized agents without a grand overview seeking to fulfill their individual needs, by negotiating with other supervisors and the service department, also known as D-planners. Although a grand plan is made by B-planners, the detailed schedules are designed decentralized by C-planners.

The OSS domain is complex, very comprehensive and the environment is unpredictable, thus transportations are associated with deviation. Therefore we will limit this thesis project to concern the daily coordination of internal transportation of heavy ship blocks at OSS, that requires KAMAG vehicles.

An obvious approach to the problem would be seeking to describe the problem in terms of agents, thus this thesis will seek to solve the logistic problem at OSS by proposing a solution strategy based on multi-agent terminology.

This thesis seeks to investigate if a MAS can be used to solve the logistic problem at OSS. We will investigate the MAS terminology, propose a solution strategy and implement basic concepts from the solution strategy in a multi-agent system.

Furthermore we will make a 3D simulation model of the OSS domain to give decision-makers the posibility of identifying the consequences of their actions including identifying transportation bottlenecks.

As interface between the multi-agent system and the simulation model, we will design a middleware application. The overall system can be seen in figure 2.15.

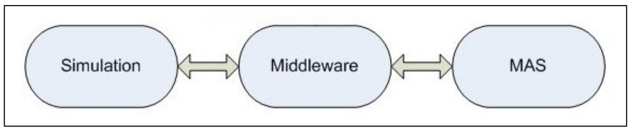


Figure 2.15: The overall system

2.5.1 Objectives

The major objective in this thesis is to design a decision support tool for coordination of the daily transportations of ship blocks at OSS, based on emulation and multi-agent technology.

The objective can be categorized into three objectives:

1. **Emulation:** Design a simulation model of the OSS domain.

2. **Multi-agent system:** Design a multi-agent system using a MAS framework to handle the logistic planning.

3. **Middleware:** Design a middleware application to exchange data between the simulation model and the multi-agent system.

2.6 Project Workflow

We began this project researching the OSS domain and gathering information, interviewing several people, such as B-planners, C-planners and D-planners, which gave us an understanding of the current situation residing in the problem domain. We have used the collected information to make a demarcation of our project, and specified our main focus areas.

In this thesis we will go through several phases as seen in figure 2.16.

In the analysis phase we will investigate the multi-agent terminology, research possible suitable simulation tools, and find out basic knowledge regarding middleware applications.

In the design phase we will design a solution proposal in terms of multi-agent technology, a simulation model of the OSS domain and finally a middleware application connecting our multi-agent system with the simulation model.

In the implementation phase we will implement the three subsystems: the simulation model, the multi-agent system and the middleware application.

In the test phase we test subparts of the overall system and describe their final status.

A journal concerning interviews, meetings and presentations can be found in appendix J.

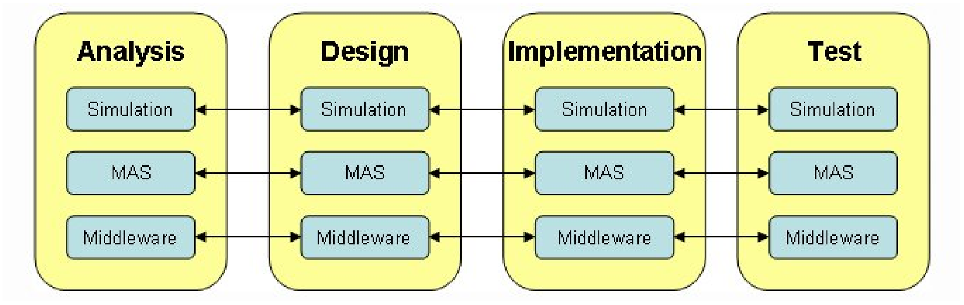


Figure 2.16: Project Workflow

Chapter 3

Related Work

This chapter presents projects, which is related to our thesis project in one way or another. For each related project we briefly describe the objectives and content, and finally we summarize how the projects are related to our thesis project in a summary section.

3.1 Coordinating Mutually Exclusive Resources using GPGP

The article in [17] approaches a hospital scheduling problem by proposing a multi-agent solution using Generalized Partial Global Planning or GPGP(described in section 8.5), that preserves the existing human organization and authority structures. The multi-agent system increases hospital unit throughout and decreases patient stay time.

The hospital scheduling problem is represented with TAEMS (described in section 7.2), which is a language for structuring tasks. The tasks that have to be solved, either have the status "done" or "not done", meaning that a task is either completed or not. All tasks must be completed, and there are no deadlines referring to when a tasks must be completed. Travel times for patients are not represented in the hospital scheduling problem.

An extension of GPGP with a coordinated mechanism handles mutually exclusive resource relationships. This new mechanism can be applied to any problem with the appropriate resource relationship, like the other GPGP mechanisms. The effect of increasing interrelations between tasks performed by different hospital units are examined with the new mechanism and the mechanism itself is evaluated in the context of the patient scheduling problem.

A resource manager agent is implemented, which centralizes a bid processing among agents, where the agent with the lowest bid is selected to perform the task, but a multi-stage negotiation, where conflicts are bounced back at the agents for further negotiation, is investigated.

3.2 The Advanced Logistics Technology project

It was in 1996, that DARPA[1] started the advanced logistics project(ALP) with a 80 million dollar budget, aimed at developing the next generation of logistics systems [6]. The DARPA advanced logistics project investigated and demonstrated technologies which are expected to make an important contribution in future transportation and logistics.

Logistics concerns getting the right "stuff" to the right place at the right time, and major transportation vendors are beginning to require the efficient solution on a minute by minute basis, likewise major coorporate organizations such as GM and Ford, needs to support a smooth inflow of production materials as well a smooth outflow, thereby requiring an optimal inventory.

The next generation of logistics systems will be information systems, that manipulates massive, distributed, logistics databases, and tracks the status changes of supplies and resources, and furthermore replans as needed in order to accomplish missions at hand.

The ALP primarily concerns the following three main areas:

1. **Rapid Supply:** Finding rapid supply is of crucial importance for every major organization in order to keep the business running smoothly. Companies must rely on their suppliers to deliver on time in order to satisfy their own deadlines to their customers, for instance, an Internet bookseller would not be able to supply books for customers if it were not able to rapidly find suppliers. The US military could not function, if it were not able to locate supply vendors for food and ammunition, hence rapid supply is an important part of the next generation logistics systems.
2. **End-to-End Movement Control:** Once suppliers have been found and has commited to supply, the movement of supplies is considered to be another major task for most organizations. Today many organizations use a lean inventory approach, which requires carefully coordination between suppliers, customers and transportation vendors.

[1]Defence Advanced Research Projects Agency

3. **Execution Monitoring:** Every optimal plan is destined to undergo changes in real world application, since plans frequently go wrong when supply and movements plans are in place, hence monitoring the state of affairs in order to ensure compliance with the planned states, is crusial for companies and military organizations. This indicates a requirement for monitoring the state of supply continuously, detecting deviations, and corrective actions conducted in a timely manner.

The objectives of the ALP project is to propose description solutions to the following cases:

1. How to locate available materials
2. How to locate available transportation resources
3. How to locate materials needed to perform a given task
4. How to handle multiple requests to same resource
5. How to monitor state changes
6. How to perform dynamic replanning in a scalable way
7. How to monitor the global status of the supply chain

Figure 3.1 shows the system processes requests; first the cluster interface passes a request to the Task Expander component. The Task Expander creates a set of tasks, which must be solved in order for the initial request to be solved, and creates an OR-graph as seen in figure 3.2. If we consider the example of sending 50 units of some item "1928282" from Charleston, SC to Toslic, then we first start by flying from SC to Tuzla, from where we can either use heavy Equipment Transporters(HET) to transport the items from Tuzla directly to Toslic, or we can use trucks to transport the items from Tuzla to Braz, and further from Braz to Tozlic. This example demonstrates that different paths in the OR-graph corresponds to different sequences of actions that can be used to satisfy a request. Next, the Allocator component allocates resources to the task, and computes a set of possible solutions, which are computed in multiple stages. The Allocator then selects the solutions with the least cost. When the Allocator component has succesfully done its part, the Assessor component takes over. The Assessor monitors the generated plan to see if it is on schedule and takes action if the schedule deviates from the plan.

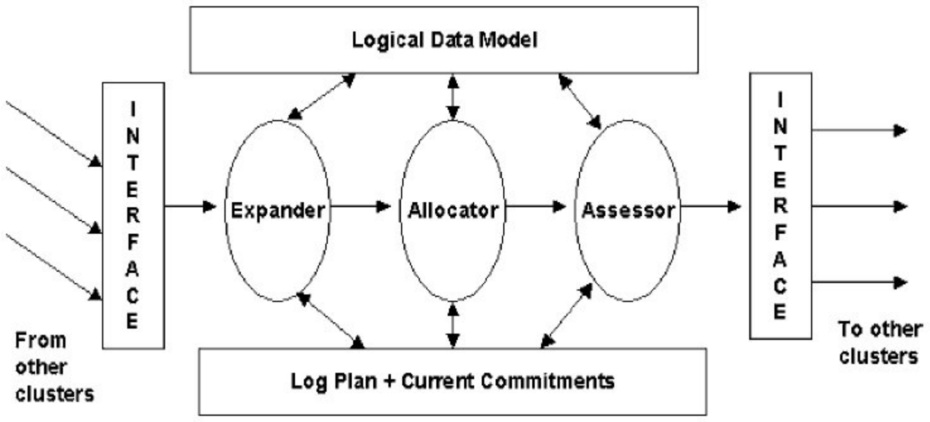


Figure 3.1: Architecture of an ALP Cluster

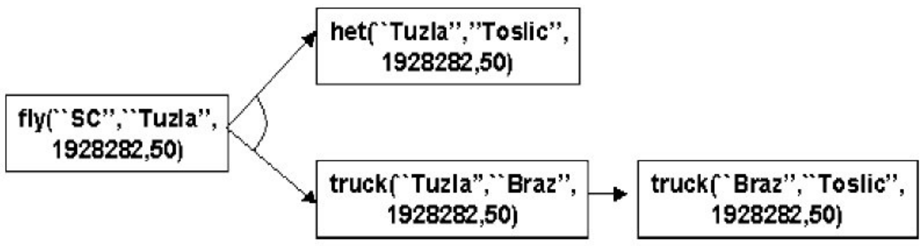

fly(``SC'',``Tuzla'', 1928282,50)
het(`Tuzla'',''Toslic'', 1928282,50)
truck(`Tuzla'',``Braz'', 1928282,50)
truck(`Braz'',``Toslic'', 1928282,50)

Figure 3.2: Example of an expanded task OR-graph

3.3 The Ultra*Log program

In 2001 DARPA[2] started the Ultra*Log program that corncerned further development and maintance of the Cougaar Agent Architecture that was developed for DARPA under the advanced logistics project.

The main objectives of the Ultra*Log program have been the following[3]:

- Extend and enhance the Cougaar capabilities to achieve the UltraLog project goals for new robustness, scalability, stability and security capabilities in a highly chaotic operational environment.
- Generate a set of software products and facilitate their use by the UltraLog community for software design, development, testing and integration.

[2]Defence Advanced Research Projects Agency
[3]http://dtsn.darpa.mil/ixo/programs.asp?id=61

Figure 3.3: Example of military network being compromised

- Evolve an agent-based architecture and requisite supporting material to provide a leave-behind capability and enable technology transition to a broad operational community

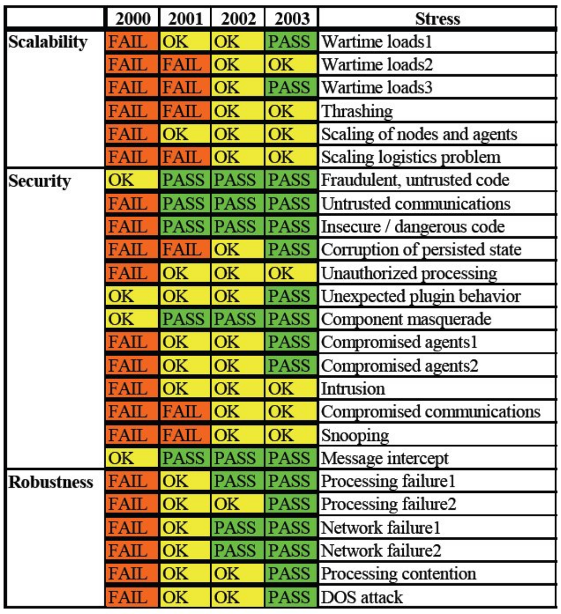
From the above mentioned objectives the following were adressed, and the results are shown in figure 3.4 [30]:

- Robustness: A Cougaar application should survive the loss of any individual components and/or hardware substrate with minimal loss of functionality. This includes automatic recovery of lost agents, as well as various mechanisms to conserve resources and to use redundancies efficiently.
- Security: A Cougaar application should be capable of repelling various sorts of electronic attacks, should maintain information integrity, and should avoid exposing communications as much as possible.
- Scalability: The Cougaar infrastructure should not have any intrinsic scalability issues. It should be possible to implement Cougaar applications which scale to the degree that the application logic allows.

The Ultra*Log program seeked to research, develope and demonstrate a prototype with a society consisting of more than 1000 agents of medium complexity. The goal was to make the system robust to changes in a chaotic environment (highly dynamical), that is 90% as chaotic as the most real world environment, such that it should be operative with a 45% information infrastructure loss, with no more than a 20% capability degradation, and no more than a 30% performance degradation. Figure 3.3 illustrates a military network being compromised.

The result of the Ultra*Log program in 2001 was an increasing planning speed by 4x over ALP(section 3.2), better schedules, inventory managers, and better use of multiple fidelities and sliding time windows, more parallelism. The system verified continuous operation under kinetic attack and simultaneous failure of 40% [28]. The system proved to be robust to changes, by detecting failures, allocating new resources and restoring functionality from denial of service attacks.

From 2002 the Ultra*Log worked on making the system survivable, since this was not improved much in 2001.



	2000	2001	2002	2003	Stress
Scalability	FAIL	OK	OK	PASS	Wartime loads1
	FAIL	FAIL	OK	OK	Wartime loads2
	FAIL	FAIL	OK	PASS	Wartime loads3
	FAIL	FAIL	OK	OK	Thrashing
	FAIL	OK	OK	OK	Scaling of nodes and agents
	FAIL	FAIL	OK	OK	Scaling logistics problem
Security	OK	PASS	PASS	PASS	Fraudulent, untrusted code
	FAIL	PASS	PASS	PASS	Untrusted communications
	FAIL	PASS	PASS	PASS	Insecure / dangerous code
	FAIL	FAIL	OK	PASS	Corruption of persisted state
	FAIL	OK	OK	OK	Unauthorized processing
	OK	OK	OK	PASS	Unexpected plugin behavior
	OK	PASS	PASS	PASS	Component masquerade
	FAIL	OK	OK	PASS	Compromised agents1
	FAIL	OK	OK	PASS	Compromised agents2
	FAIL	OK	OK	OK	Intrusion
	FAIL	FAIL	OK	OK	Compromised communications
	FAIL	FAIL	OK	OK	Snooping
	OK	PASS	PASS	PASS	Message intercept
Robustness	FAIL	OK	PASS	PASS	Processing failure1
	FAIL	OK	OK	PASS	Processing failure2
	FAIL	OK	PASS	PASS	Network failure1
	FAIL	OK	PASS	PASS	Network failure2
	FAIL	OK	OK	PASS	Processing contention
	FAIL	OK	OK	PASS	DOS attack

Figure 3.4: Improved stress result spanning from 2000 to 2003

3.4 DVMT

The Distributed Vehicle Monitoring Testbed(DVMT) was developed at the University of Massachusetts. The work on DVMT began in 1981 and continued until 1991. DVMT is one of the oldest and largest distributed AI testbeds [2] and has resulted in over 50 papers and PhD theses in many areas.

The DVMT simulates a network of vehicle monitoring nodes (agents). DVMT operates in an environment, where sound sensors are placed to cover the

environment. Each node is associated with one or more sensors and is responsible for analyzing accoustically sensed data to identify, locate and track patterns in the environment, which is mapped into a 2-dimensional map.
An example from [13] is shown in figure 3.5, where two agents coorperates and finds the track for a vehicle in the environment. A brief description of the figure is given beneath the figure. At the figure to the left in

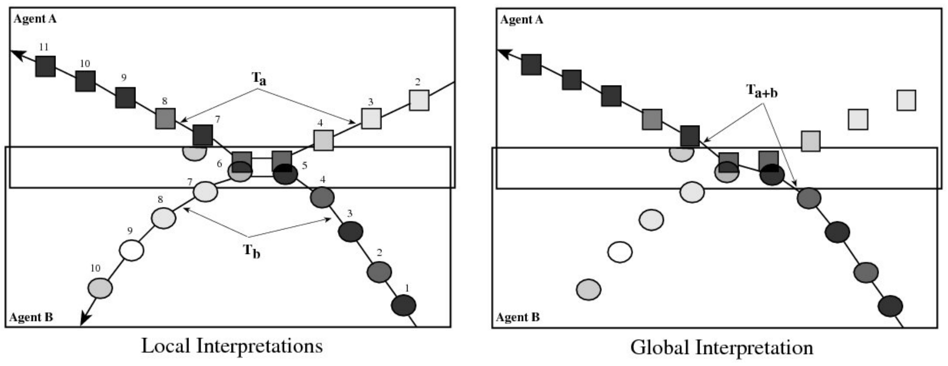


Figure 3.5: An example of inconsistent local interpretations

figure 3.5 two rectangels are shown, one for agent A and one for Agent B. Each rectangle represents the part of the environment that the agents are observing, and in this case the environment that the agents observe overlaps. The squares represents a vehicle that has been observed by agent A , and the circles a vehicle that has been observed by agent B. The grey density in the squares/circles represent the "quality" of the location measurement of the vehicle; the darker density the better quality. At the left figure agent A locally beliefs that a vehicle has moved on the track T_a at the timesteps from 2-11 and agent B locally beliefs that a vehicle has moved on the track T_b at the timesteps from 1-10. This is though not possible, because in this case either two vehicles has to be at one location at the same time or one vehicle has to be at two locations at the same time. So when the agents exchange knowledge and interact, they will coorperate and by taking the quality measurements in account, they will have a global interpretation saying, that a vehicle has moved on the track T_{a+b} shown at the right figure in figure 3.5.

3.5 ACROSS

The Agent Complex Reasoning Simulation System(ACROSS) is a scenario where a logistic problem in a non-collaborative environment with self-interested agents is solved. Agents that are part of the scenario have no common goals and their cooperation is typically financially motivated [15].

The agents in ACROSS has three types of information, namely:

1. *Public Information* is available to all agents in the system. It includes the agent identity, services proposed to other agents and other relevant characteristics it wishes to reveal.
2. *Semi-private Information* is the information which the agent agrees to share with selected partners in order to streamline their cooperation.
3. *Private Information* is available only to agent itself. It contains detailed information about agent's plans, intentions and resources.

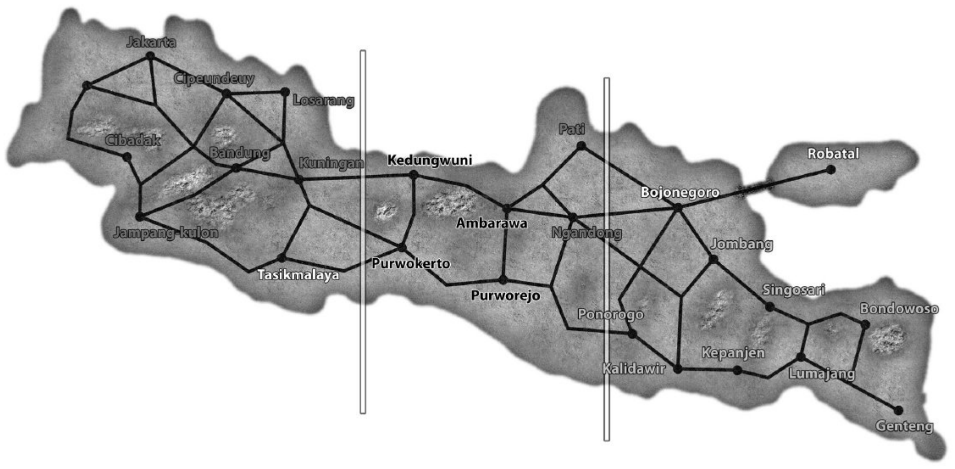


Figure 3.6: ACROSS scenario. The geography of the island is modelled after the real Java island in Indonesia, with necessary simplifications.

The environment in the ACROSS scenario is based on a 3D-model of the Java island from Indonesia as seen in figure 3.6. Three main types of agents exists in this environment:

1. **Location agents** represents the villages (see figure 3.7) in the island. Each location agent can create, transform or consume resources, but may need extra or other kinds of resources. In this case the location agent (representing the village), will start a sealed bid auction, where other location agents can bid[4] if they are in posession with the resources needed. When the negotiation is over, the buying location agent contacts coalitions of transporter agents to carry the cargo.

[4]The auction setup is done according to the FIPA Contract Net Protocol, see section 5.7.2

2. **Transporter agents** have the driver agents (see figure 3.8) as their resources. Whenever a location agent requests a transportation, then any transporter agent, which doesnt have enough driver agents to carry the cargo, will try to make a one-time coalition with another transporter agent. This is because all transporter agents are self-interested and don't wish to cooperate with all other transporters. They only pick the partners that are compatible with their private preferences. The compatibility is checked using the public information available about the potential partner and agents' private preferences.

   Transporter agents can make alliances[5], to minimize the time to create coalitions. By this way the transporters can relatively fast and efficiently submit their bid before timeout elapses.

3. **Driver agents** drive the vehicles (see figure 3.8) owned by Transporter agents. They handle path planning, loading, unloading and other driver duties.

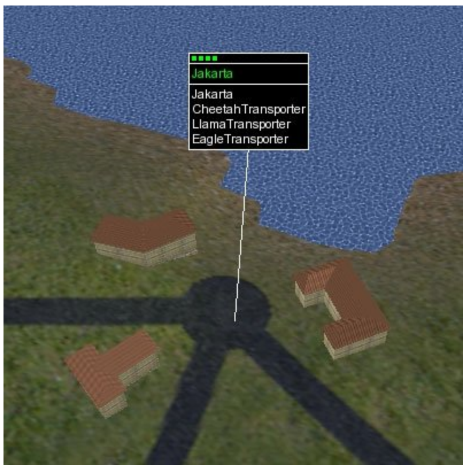


Figure 3.7: An ACROSS location agent and three transporter agents

Figure 3.8: An ACROSS driver agent

3.6 Summary

To summarize, this chapter presents work done by others, which have some relevance to our project.

[5]Alliances are groups of agents who agree to exchange the semi-private information about their resources.

The hospital system uses GPGP to solve a logistic problem at a hospital, where the essence is to coordinate and increase patient throughput, and with reference to our project we can use the same approach where we have to find space for ship blocks in storage locations, instead of finding beds for patients. The distinction between the hospital scheduling problem and our transportation scheduling problem is that our transportations have a deadline referring to the time a ship block must be picked up at a given location.

The ALP project is highly related to our thesis project, but we do not need to consider rapid supply in our thesis project, since C-planners(see description in section 2.2.3) in our project already have found suppliers, and are only interested in a transportation service, hence the end-to-end movement control and execution monitoring are related to our project. The transportaion of ship blocks needs to be coordinated, and the monitoring of the execution is important in order to give feedback to planners and dynamic replan the coordination of transports.

The Ultralog project is a related project, which has proven that using multiagent systems is useful for developing complex systems that are robust to changes in chaotic environments. This program tested a system with more than 1000 agents of medium complexity, solving major military logistic problems, wherin they were allocating resources, coordinating allocations, and replanning dynamically. The program showed that multi-agents system are highly functional even under network compromising.
The ACROSS scenario is a logistic system implemented with the A-globe framework[6]. ACROSS contains three kind of agents to coordinate transports of resources from villages to other villages in an island. These agents are related to the following actors at OSS in our case:

- Location agents in ACROSS are related to the C-planners at OSS, because it is those agents that requests transportations. The difference is that location agents makes sealed auctions for transportation tasks, whereas the C-planners requests transportations directly from the D-planner.

- Driver agents in ACROSS are related to the KAMAG vehicles at OSS, because it is those agents that handles path planning, loading, unloading and other driver duties.

The DVMT monitors vehicles in an environment. It does this by using sound sensors which are placed all over the environment. By this way the vehicles can be tracked in the environment, and their current locations can be found. The DVMT project is related to our project in the sense that it

[6]see section 5.9.1 for information about A-globe.

could be used to navigate the KAMAG vehicles around at the OSS domain. Currently the KAMAG drivers are planning their routes themselves, but by using the DVMT approach, the vehicles could have been navigated by the multi-agent system instead.
The related work presented here have similar problems used in other context, which states several methods which could be interesting to use and we will explore the possibilities of involving some of the methods to solve the comprehensive logistic problem at the Odense Steel Shipyard Group.

Part II

Analysis

Chapter 4

Simulation

Simulation is used to simulate reality, but what is simulation? We will describe the concept of simulation briefly and further describe ten different simulation tools. We finalize this chapter by comparing the ten simulation tools for usefulness in respect to this thesis project.

4.1 What is Simulation?

A simulation is an imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system[1].

"For Distinction Sake, a Deceiving by Words, is commonly called a Lye, and a Deceiving by Actions, Gestures, or Behavior, is called Simulation."

— Robert South (1643-1716)

A computer simulation is an attempt to model a real-life situation on a computer so that it can be studied to see how the system works. By changing variables, predictions may be made about the behavior of the system.

Key issues with simulation modeling involves selection of key characteristics and behaviours and using simplifying approximations and assumptions within the simulation and validating the simulation outcome. Simulations are today used in many contexts, such as human systems to gain insight into their functioning. The simulation technology is likewise used in context involving such things like safety engineering, training(e.g. pilots), testing, performance optimization(e.g. manifactoring system) and education. Strong benifits with simulation models is the ability test alternative

[1]http://en.wikipedia.org/wiki/Simulation

conditions and courses of actions.

Simulation models are typically categorized into the following three types [19]:

- **Continuous time:** is one whose state varies continuously with time; such systems are usually described by sets of differential equations.
- **Discrete time:** in a discrete time system, only selected moments in time are considered, which are usually evenly spaced. Change of states are only observed at observation points. A continuous time system can be simulated by choosing a suitable small interval between observation points.
- **Continuous time-Discrete Event:** In discrete event simulation, the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system. The operation path is completely determined by the sequence of event times (which need not be evenly spaced and can be of arbitrary increments) and by the discrete changes in the system state which take place at these times (i.e., the interactions of the events). In between consecutive event times the system state may vary continuously.

4.2 Simulator requirements

In this section we describe the capabilities that should be included in the simulation tool used to model the OSS domain.

1. **Graphical 2D or 3D:** Since we were interested in modeling the Odense steel shipyard in respect to the internal transportation of ship blocks, we need a simulation tool that can represent the OSS domain in a graphical environment(2D or 3D).

2. **External access:** A strict requirement is external access to the simulation model, due to the fact that we are going to control the simulation model at runtime from another system.
3. **Ability to model reality:** The simulation model should be as realistic as possible(ability to model reality), thus the degree of graphical complexity that the simulation model can provide will have a big influence on our decision.
4. **Well documented API:** It is important that the simulation tool has a well documented API, simplifying programming tasks.

5. **Well documented:** The simulation tool should be well documented in general, so that we quickly are able to identify the tool features, capabilities and limitations.

6. **Success:** the simulator should have been used succesfully in other projects before.

7. **Statistics:** This capability should be provided, so we dont have to run the simulations at real-time, but are able to speed up the process and check the results after completion.

8. **Category:** we will note if the simulation tool, is based on discrete time, continuous time or continuous-time discrete event.

9. **Kinematics:** the ability to model kinematics can come in handy if we are going to simulate the lifting and placing process of a KAMAG vehicle.

10. **Survivability:** The simulator should be surviable, meaning that it is likely that the software provider will be on the marked in the future.

11. **OO:** The simulator should be object-oriented, allowing flexibility for the developer when designing the model.

12. **Java:** It is preferred that the simulator is developed in Java.

13. **Stable:** The simulation tool should be a stable release, i.e no alpha or beta versions.

14. **Support:** It is preferred if the tool provider offers support in case of problems with the tool.

15. **Free:** It is preferred if the simulator is free.

to simulate the OSS domain in respect to their transportation problem. The summary section that succeeds this section collates the ten simulation tools with respect to the requirements we described in section 4.2.

4.3 Simulation tools

In this section we describe ten different simulation tools, which we have briefly investigated in order to find a suitable simulation tool. The simulation tool should be used to design a simulation model of the Odense steel shipyard. We want to visualize the OSS domain graphically, either 2D seen from bird perspective, or 3D viewable from any angel.

4.3.1 AutoMod

AutoMod is a state of the art software simulation solution, that is truly useful as a decision making tool. It provides a suite of tools, which can be used to build highly accurate models for analysis, development, as well as control system emulation[2]. A screenshot of an automobile simulation model is shown in figure 4.1

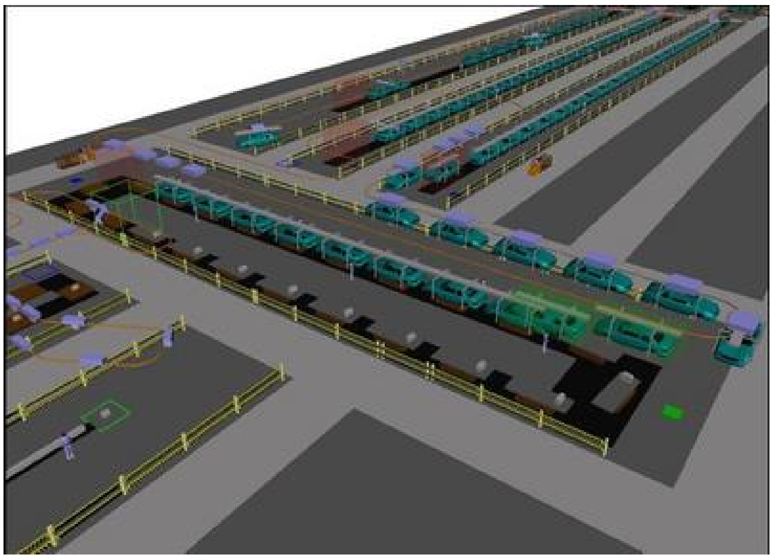


Figure 4.1: AutoMod simulation model of an automobile factory

AutoMod is different from other simulation tools in the sence that there are no limitations in respect to model size, complexity, or level of detail for operational rules. In AutoMod a template is provided, that assures accurate modeling of material movement, such as conveyors, lift trucks, operators, overhead cranes or automoated vehicles.

A true scale 3-D virtual reality animation is provided by AutoMod, which helps validating the model and communicate the design visually. CAD-like features are used to define the physical layout of manufactoring, material handling

The set of material handling systems provided by AutoMod have been developed with real-world experience in industrial automation.

In order to use AutoMod one must obtain a license, which is not free, but

[2]http://www.brookssoftware.com/pages/245_automod_overview.cfm

with a runtime license users can test "what-if" scenarios on a pre-built model.
AutoMod is a discrete-event based simulation tool[3].

4.3.2 Dymola

Dymola is a simulation tool based on the OO language Modelica, that contains a standard libary for electrical, rotational and translational mechanics, thermal components and continuous discrete input/output blocks[4]. The Modelica translator is able to perform all necessary symbolic transformations for large systems (> 100 000 equations) as well as for real time applications[5]. A robot from the MultiBody library is shown in figure 4.2.

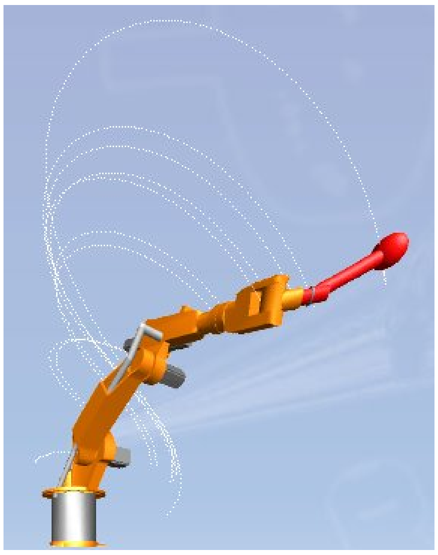


Figure 4.2: A robot simulated in Dymola

Dymola has unique multi-engineering capabilities meaning that models can contain components from various engineering domains, allowing the complete system better to model reality. Moreover a graphical editor with various libaries is embedded with Dymola. An interface to Simulink is also provided.

Dymola can handle algebraic loops an reduce degree-of-freedom caused by constraints, using symbolic manipulation, thus making it robust and

[3]http://en.wikipedia.org/wiki/Simulation_language
[4]http://www.dynasim.se/models.htm
[5]http://www.modelica.org/tools

giving it high performance. Furhtermore special numerical solvers enables real-time HILS[6].

A demo version of Dymola can be downloaded from dynasims website[7].

4.3.3 GoldSim

GoldSim is a highly graphical, object oriented, general purpose simulator for nearly any kind of physical, dynamic, financial or organizational system. One can view GolSim as visual spreadsheet, wherein users can visually create and manipulate data and equations [39]. GoldSim supports management and decision making in business, engineering and science.

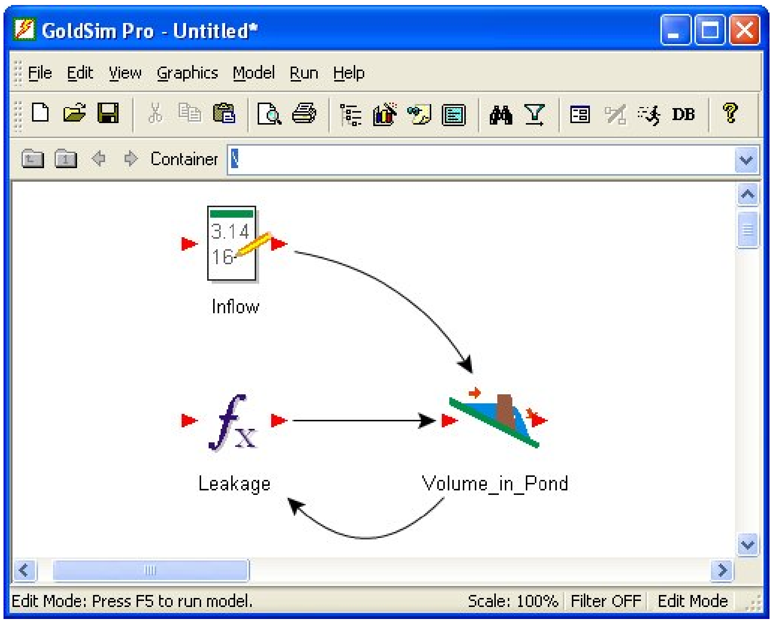


Figure 4.3: Screenshot of GoldSim

Objects constructed in a GoldSim model are referred to as elements, which represent building blocks of the model. Each object has a grahical image attached to it, and they accept input data and produces output data.

GoldSim is specially designed to address the uncertainty that is present in real-world systems, and provides tools for representing uncertainty in processes, parameters and future events. Likewise the occurence of discrete

[6]Hardware-In-the-Loop Simulation
[7]http://www.dynasim.se/demo/

events into continuous varying systems, and the construction of large, complex models, are supported.
GoldSim is a commercial simulation tool, but is offered free for students, teachers and professors.

4.3.4 JavaSim

JavaSim is an object-oriented discrete-event based simulator. This simulation tool is based on the Java programming language, and includes statistics.

JavaSim provides a user manual describing its various classes, but this manual is not considered to be well documented (unsufficient explaination to the various classes, empty example sections, ect.). This simulation tool was last tested on JDK 1.0.2 and 1.1.x on Solaris, linux, and Windows 95/NT 4.0 and the documentation have not been updated since 1999, thus this simulation tool is not considered to be survivable.

We have not found any documentation stating that this simulation tool have been used succesfully in other projects. We have not been able to find documentation confirming that this tool can be used to design a graphical environment like the OSS domain, other than a briefly described draw function.

4.3.5 Micro Saint

MicroSaint is a object-oriented general purpose simulation tool, based on discrete-events, that uses a flow chart approach for modeling, and have been proven useful for both small businesses and fortune 500 companies. MicroSaint have been used in military projects, health care, manufactoring, the service industry and in human factors[8]. A screenshot of Micro Saint is shown in figure 4.4

The flexibility, power and tools for optimazation makes MicroSaint a useful tool for answering "what-if" questions, and is useful for simulating systems of any size, shape and complexity.
There three types of views, which are 2D animation, 3D animation or network diagram view.

With Micro Saint we can create 3D models for represented the environment, and the enhanced viewing capabilities makes it possible to rotate and/or view 3D objects from any angle. Micro Saint uses Microsofts DirectX technology. The C# programming language is applied in this simulation tool.

[8]http://www.maad.com/index.pl/micro_saint

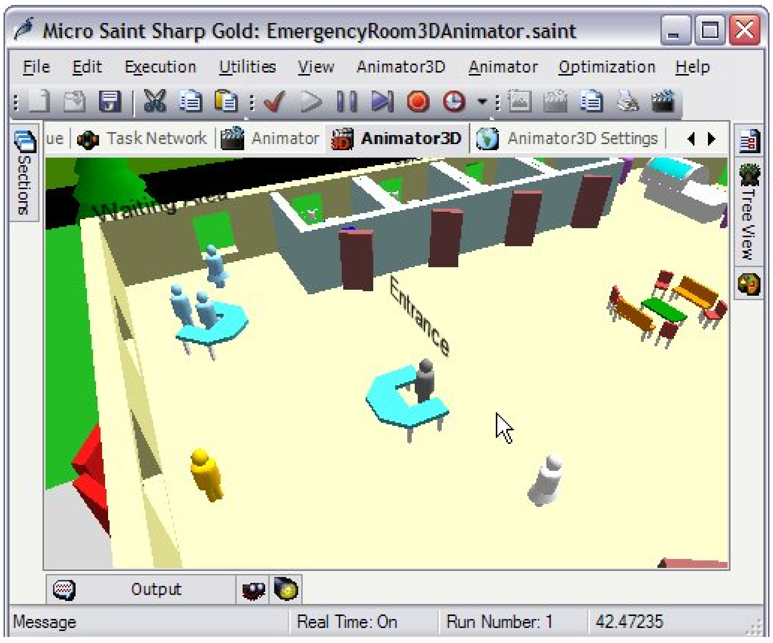


Figure 4.4: Screenshot of Micro Saint

4.3.6 Simcad Pro

Simcad Pro is a a discrete-event based process simulation software for manufacturing, automation, distribution and logistics. It includes a fully dynamic environment with optimization and lean and six sigma support[9].

It is possible design, validate and implements ideas without disturbing current operation. An overall process flow is defined at the beginning, and then each process cell can be expanded into individual processes of its own.

A graphical user interface is provided, that enables the creating of simulation models without haven to write a line of code, meaning that non-programmers are able to use this simulation tool.

Models can be verified graphically through animation. Simcad Pro can be controlled from an external application or can interact with external tools for data transfer and control.

Simcad Pro is the only graphical tool that enables model modifications during the simulation run. Optimization can be performed through the built-in

[9]http://www.createasoft.com/processImprovementSimulator/leanProcessSimulationSoftware/SimcadProProcessSimulator7.2.html

optimizer or by dynamically interacting with the model. Changing object flow and modifying constraints is done while the simulation is running. Simcad Pro eliminates the need for warm up period required by other tools and enables real time optimization.

In Simcad pro directional arrows, which contain specialized behaviours, are used to model part transition, e.g. connectors can behave like conveyers with photo-eye control. Furthermore processes are capable of implementing change-over time (between products or at specific times) and mean time between failures and repairs.

Simcad Pro is a 2D and 3D visualization and animation engine, that computes the animation without user intervention. Interfaces are provided to support CAD files(dwg/dxf) and distances based on the locations of the different entities are computed automatically. Furthermore the 3D environment provides an improved visualization for high end presentations and dynamically extracts all values to the 3D views, therefore reducing the amount of work required to build 3 dimensional simulations. A screenshot of a 3D model is shown in figure 4.5.

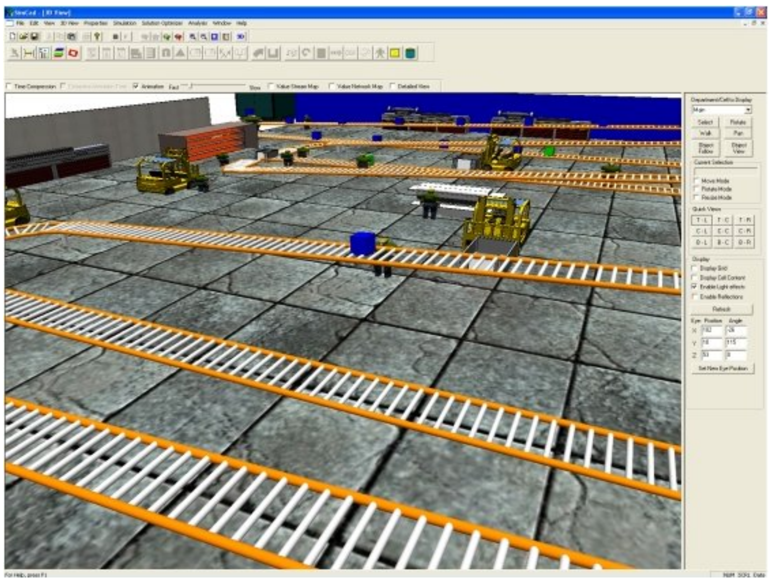


Figure 4.5: Screenshot of Simcad Pro

4.3.7 SimCreator

SimCreator is a graphical simulation tool based on real time simulation and modeling. An intuitive GUI allows users to choose and connect components to build models in a power flow modeling approach [44]. A screenshot of the Driving Simulator Model is shown in figure 4.6

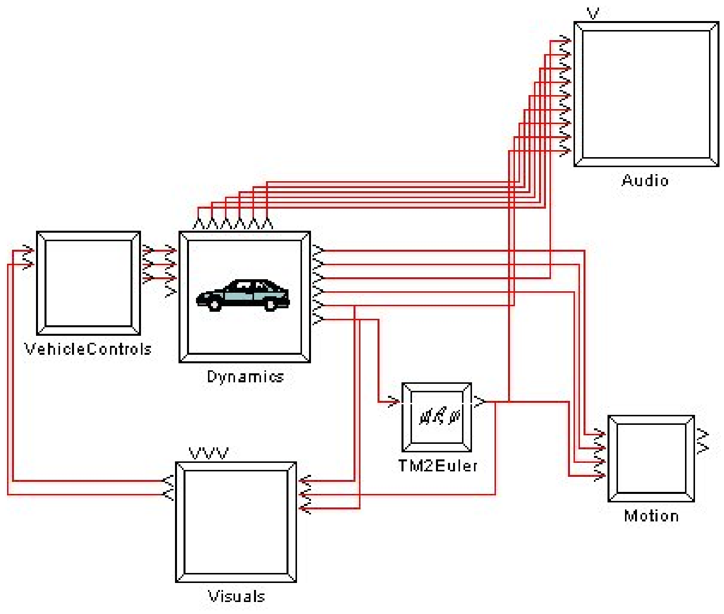


Figure 4.6: Screenshot of SimCreator

Models developed with SimCreator can be used directly in real time hardware. The code, that SimCreator generates are used in hundreds of simulators and games. Users can create distributed models, without having to write a single line of C code, by using the GUI. SimCreator is useful for developing models for simulating dynamic digital controllers and real time multibody vehicle dynamics.

SimCreator is specialized in generating code to run on embedded systems, multiprocessor and distributed systems and delivers uncompromised real time.

SimCreator provides a standard libary of components, including component libaries, that allows users quickly to create their own customized compoents fulfilling their specific needs. Furthermore components can be grouped to form other compoents. The mix of C code components and high-level modeling makes this tool useful for programmers, engineers and end users, such that they together can develope an interactive model.

4.3.8 Simprocess

SIMPROCESS is a hierarchical integrated process simulation software package that combines Process Mapping, Flow Charting, Discrete Event Simulation, and ABC[10] in a single easy to use tool[11]. Simprocess is an object-oriented tool useful for analysis and process modeling by combining the power of simulation, statistical analysis, ABC and animation. Simprocess provides non-programmers with a standard set of ready-made blocksfor building logic-based models, while supporting decisions in customers BPM[12], Process Improvement, Six Sigma and BAM[13] initiatives.

SIMPROCESS provides the most costeffective, accurate, and rapid strategic weapon for businesses to evaluate alternatives prior to implementing them. The ability to visualize how a process would behave, measure its performance, and try "what ifs" in a computer model makes SIMPROCESS an invaluable tool for making tough decisions before you commit expensive resources, time, and money.

Simprocess proposes the following four steps for modeling company processes:

1. Create model: The creating of a model involves three main tasks:
   (a) first map your processes using a flowcharting and process documentation facility(if creating a flowdiagram),
   (b) then drill down inside the hierarchical processes as seen in figure 4.7, and define subprocesses, activities and workflow(with no limitation of the number of sublevels). In this stage you also define objects.
   (c) finally resources and their usage are defined. Resources can be defined as mambers of departments and/or workgroups, and can be assigned individually to activities or processes, because they are hierarchical like processes.
2. Simulate the process: Before simulating the model, performance measures should be selected, i.e. throughput. Simprocess then provides an animated picture of the flow in the model, providing real-time graphs of performance measures.
3. Analyze the results: Simprocess generates concise reports of results containing throughput, wait-time, resource utilization and cost reports. Furthermore it is possible to generate customized reports.

[10]Activity Based Costing
[11]http://www.simprocess.com
[12]Business Process Management
[13]Business Activity Monitoring

4. Evaluate alternatives: Two unique functions called Alternatives and Scenario Manager, can be used to evaluate alternative representations.

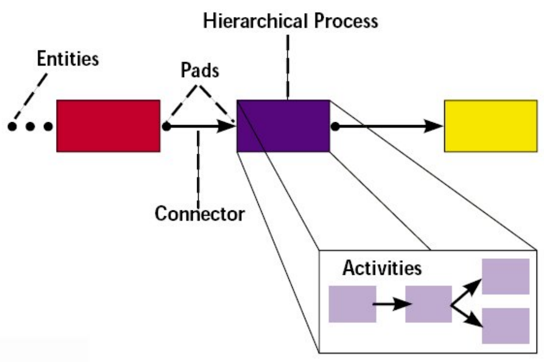


Figure 4.7: Creating models with Simprocess

It is possible to download a trial version from the Simprocess download website[14], and one can within 15min begin to evaluate this simulation tool.

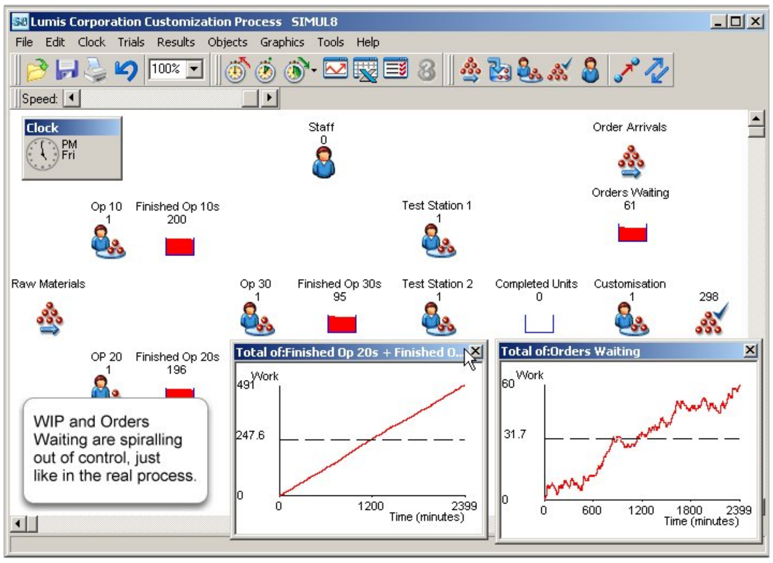
4.3.9 Simul8

SIMUL8 is commercial discrete event simulator designed to adress various business problems, including supply chain management, design of manufactoring systems and capacity planning issues[15]. SIMUL8 is a useful tool for analyzing flow, quering and general resource requirements, thus it generally used prior to the actual layout of a facility[16].

Brooks Software which is provider of both AutoMod and SIMUL8 have formed an alliance in the simulation industry, due to the fact that SIMUL8 and AutoMod(described in section 4.3.1) are very complementary simulation products. Where AutoMod excels in realistic and highly detailed modeling, SIMUL8 excels in cenceptual modeling, i.e. flow charting and BPR[17].

AutoMod is applied in the more detailed analysis stage once an initial layout of the facility is available and the specifics of material movement are determined

[14]http://www.caciasl.com/downloads/downloads.cfm
[15]http://www.simul8.com/products/features/
[16]http://www.brookssoftware.com/pages/248_simul8.cfm?searchterm=simul8
[17]Business Process Reengineering



WIP and Orders Waiting are spiralling out of control, just like in the real process.

Figure 4.8: Screenshot of a Simul8 model

SIMUL8 provides a grahical environment usable for non-programmers like AutoMod, as well as graphical animations, statistics and interfaces to commen external programs, such as Excel and Visio.

Simul8 must be considered to be a highly succesful simulation tools, with references like Jaguar, Nike, IBM, Chrysler an many other companies, which have used simulation model developed with Simul8.

4.3.10 Renque

Renque is a simulation tool designed for general purpose discrete event simulations. The high abstraction level of the concept of discrete event simulation renders its application potential extremely wide-ranging. Some common application areas of discrete event simulation are service stations such as airports, call centers and supermarkets; road and rail traffic; industrial production lines and logistical operations such as warehousing and distribution [14].

Renque provides two types of elements, passive and active elements, whose parameters can be changed. Renque is closely connected to Microsoft Excel, from where the user can import data in the model.

Active and passive server classes constitues the core of the Renque simulation engine. The active servers collect entities upstream through one or more connection links, while passive servers do not collect entities and do not dispatch them either, i.e. the passive components are used for temporary storage, while the active components contain logic for collecting and dispatching. Passive servers simply stores entities, and waits until a downstream active server collects them. When an active server collects an entity(upstream), it stores it for some delay time and then dispatches it at the downstream side; this process is repeated in cycles.

Figure 4.9 shows an example of a model simulation a shopping centre. The first active component simulates customer entry in the mall, the second active component simulates a customer shopping, the third component is a passive component simulating the customer stading in a queue waiting to pay for their shopping, and the last component is an active component simulating the paydesk.

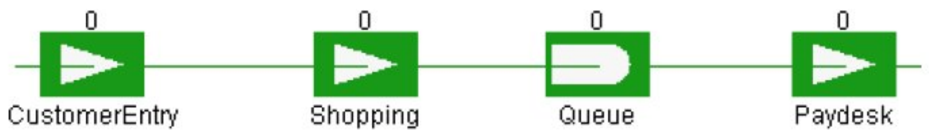


Figure 4.9: Renque simulation model of a shopping centre

The flow in the simulation model is influenced by capacity, rules and other various parameters for the active and passive servers. Passive rules include FIFO[18], LIFO[19] and random. The active rules for collect and dispatch include random, probability and order.

It is posible to download a 60 day trial version from their website[20], which gives the possibility of evaluating the simulation tool before purchasing it.

4.4 Summary

In this chapter we discussed ten simulation tools, which have all briefly been researched for usefulness in this thesis. They are all different, and can be used to design different types of simulation models with different degree of complexity an abstraction level.

[18]First In First Out

[19]Last In First Out

[20]http://www.renque.com/

The ten frameworks have been collated and compared in table 4.1 and many of our requests are generally supported by most of the simulation tools.

We have choosen to use the AutoMod simulation tool to design our simulation model of the OSS domain. This tool is based on continuous time discrete-event, and we are interested in an event based system. The tool is also well documented, has some object oriented capabilities, in respect to modeling entities in a model, and the tool is surviveable, i.e. likely to be on the marked for many years to come. AutoMod is described as one of the best simulation tools for designing detailed environments, and it is commonly used in the industry to model production line environments. All theese capabilities makes AutoMod a suitable candidate to use in this thesis project.

The AutoMod simulation has the best system to design a detailed path movement system, and also has the ability to model kinematics, which we may use in order to model the kinematics of the KAMAG vehicles at OSS. The AutoMod simulation tool is commerciel, but SimCon, which is one of the leading corparations on the danish marked, that supplies emulation solutions, are also part of the DECIDE project, and MIP[21] has therefore been able to provide us with a license key during this thesis.

[21]The Maersk Mc-Kinney Moeller Institute for Production technology
✓ = supported feature, (✓) = partly supported feature
✗ = unsupported feature, (✗) = probably unsopperted feature / Unknown

Feature / Simulator	**AutoMod**	**Dymola**	**GoldSim**	**JavaSim**	**Micro Saint**	**Simcad Pro**	**SimCreator**	**Simprocess**	**Simul8**	**Renque**
Graphical 2D or 3D	✔	(✔)	(✔)	✘	(✔)	✔	(✔)	✔	(✔)	✘
External access	✔	✔	✔	✔	✔	✔	(✘)	✔	✔	(✘)
Ability to model reality	✔	✔	✔	✘	✔	(✔)	(✘)	✔	✔	✘
Well documented API	✔	(✔)	✔	✔	(✔)	✔	(✘)	✔	✔	✘
Well documented	✔	(✔)	✔	✘	(✔)	✔	✔	✔	✔	✔
Success	✔	✔	✔	✘	(✔)	✔	✔	✔	✔	✘
statistics	✔	(✔)	✔	✔	✔	✔	✔	✔	✔	✔
Discrete time	✘	(✔)	✘	✘	✘	✘	✘	✘	✘	✘
Continuous time	✘	(✔)	✘	✘	✘	✘	✘	✔	✘	✘
CTDE	✔	(✔)	✔	✔	✔	✔	✘	✔	✔	✔
kinematics	✔	✔	(✘)	(✘)	(✔)	✔	(✘)	✘	(✘)	✘
Surviveability	✔	✔	✔	✘	✔	✔	✔	✔	✔	(✘)
OO	(✔)	✔	✔	✔	✔	(✔)	✘	✔	(✔)	✘
JAVA	✘	✘	✘	✔	✘	✘	✘	✔	✘	✘
Staeble	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Support	✔	(✔)	✔	(✘)	✔	✔	✔	✔	✔	✔
Free	(✔)	(✔)	(✔)	✔	✘	✘	✘	(✔)	✘	(✔)

Table 4.1: Comparing simulator features

Chapter 5

Multi-Agent System

Every MAS paradigm is based on agents, which communicates in order to reach their individual goals and/or global goals [32]. Agents have perception, which determines how they perceive the environment they are part of, and holds mechanisms, that gives them the ability to act according to certain situations. The agents ability to make decisions have been influenced from AI. Systemic, which focuses on the flow of information between components, have influenced the terms of interaction and communication between agents.

5.1 What is an Agent?

There exist much debate and controversy on the subject of a universal definition of an agent, but there exist no standard definition at the present time. It is important with a definition in order not to loose focus of ones objective, thus we have adapted the folowing definition from [52].

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

Autonomous agents are capable of making independent decisions, by taking actions in order to satisfy internal goals, which is partly based on their perceived environment[1].

If we take a look at figure 5.1, which illustrates an agent in its environment, it is seen that the agent perceives the environment and reacts by performing actions that will effect the environment. In most cases the agent will

[1]http://en.wikipedia.org/wiki/Software_agent

not have complete control over its environment, meaning that it at best will have a partial control in the sence that it can influence the environment.

From the agents point of view it can perform the same action twice, which could result in different outcomes(non-determinism) depending on the current situation, thus the agent should be prepared that its action might not have the effect it desires.

Most likely an agent will have a set of different actions it can perform, which represents the agents capability. The key problem for the agent is to decide which actions to perform in order satisfy its design objectives.

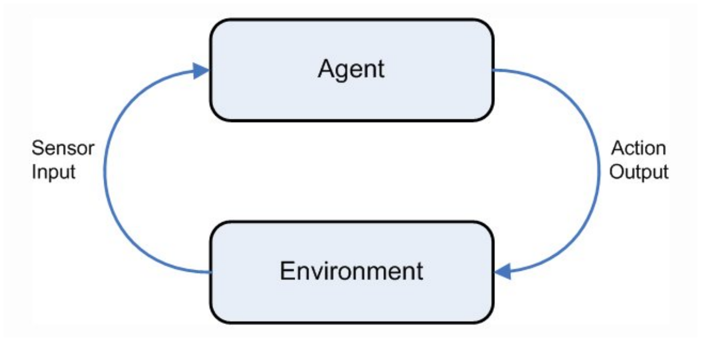


Figure 5.1: An agent in its environment. The agent takes sensory input from the environment and produces as output actions that effect it.

We will give two examples of agents, a physical agent and a virtual agent.

Example: Thermostat

A simpel control system like a thermostat could be viewed as an agent. It perceives its environment with sensors, which indicates whether the temperature is too cold or ok, and produces two signals(actions) "heat on" or "heat off" correspondingly. Like we mentioned earlier, the same action might have different outcomes depending on the given situation, if for instance someone left the door open, the action of turning the heat on might not have the desired effect on the environment.

Example: xbiff

Xbiff is a X Windows[2] program which monitors the users incoming mails, and notifies the user when a new mail arrives.

[2]GUI for Unix and Unix like operating systems

> Xbiff can be viewed as a software agent, which performs actions to indicate whether or not there are unread messages. While our thermostat example inhabited a physical environment, the Xbiff program inhabites a virtual environment, which it perceives by calling software function and performs its output actions by changing an icon for the user.

We will now describe four capabilities in which agents are frequently cathegorized, which concern reactivity, proactiveness, social ability and cognitive ability [52, 32].

5.1.1 Reactive Agents

Reactive agents have the ability to perceive their environment, detect changes and to respond to those changes in form of actions, and thereby effecting their environment as seen in figure 5.2. This type of agent has the ability to respond quickly or within reasonable time to changes in the environment, and chooses its next actions, from a predefined set of actions in its knowledge base [52, 32].

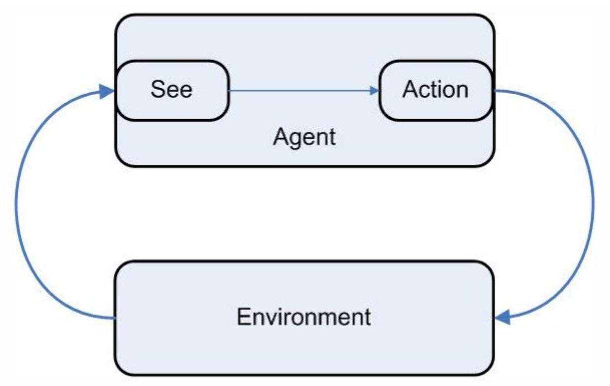


Figure 5.2: A reactive agent perceives its environment, and reacts with actions, that effects the environment.

5.1.2 Proactive Agents

Proactive agents are able to take the initiative in order to satisfy their design objectives, and thereby exhibit goal-directed behaviour [52]. The proactive agent is goal directed, and keeps an internal state (see figure 5.3), which contains its goals and set of actions, and it must be able to reason about which actions it can use in order to reach its goal. If the agent is purely proactive, then it will not react to changes in the environment until it has reached its goal, which works fine in a functional system, which do not change under execution. In complex and real world systems , the environment is likely to undergo changes before goals are reached, thus it will not be acceptable for an agent to be purely proactive and not respond to changes in the environment before their goals are reached.

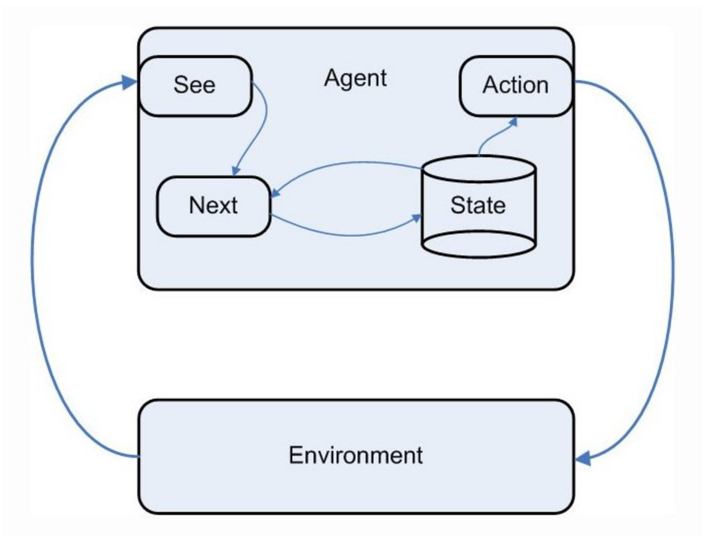


Figure 5.3: A proactive agent perceives its environment, and acts according to its goal-directed behaviour.

Below is an example of reactive vs. proactive behaviour among children [3]

> “A REACTIVE child responds to anger by throwing what most people would refer to as a tantrum. Arms may flail, feet may stomp, and tears may flow. Screaming and crying are usually a given. Actions are impulsive and emotions typically run high. A reactive pre-teen can seemingly transform into a toddler in less time than it takes to roll your eyes.”

> “A PROACTIVE child responds in a subtle, less noticeable manner. His mind begins to plot revenge against the person who has dared to "cross" him. He is calm and collected on the outside, but manipulative and deliberate on the inside. A proactive child is quite skilled at hiding his angry feelings behind an impassive expression.”

[3]http://www.parentcoachplan.com/article2.php

5.1.3 Social Agents

Social agents are capable of interacting with other agents in order to satisfy their design objectives [52]. Some tasks require collaboration between agents to reach a goal. Consider two types of agents with a common objective - to collect ore on mars; one agent with explorer capabilities and one with collector and transportation capabilities. In order for the explorer agent and the transportation agent to reach their common objective, they will have to interact; the explorer searches the planet for ore and reports to transportation agents, which then collects the ore and transports it back to a base, such agents are characterized as social agents. Figure 5.4 shows two agents, which has the ability to perceive each other, and thereby to be social.

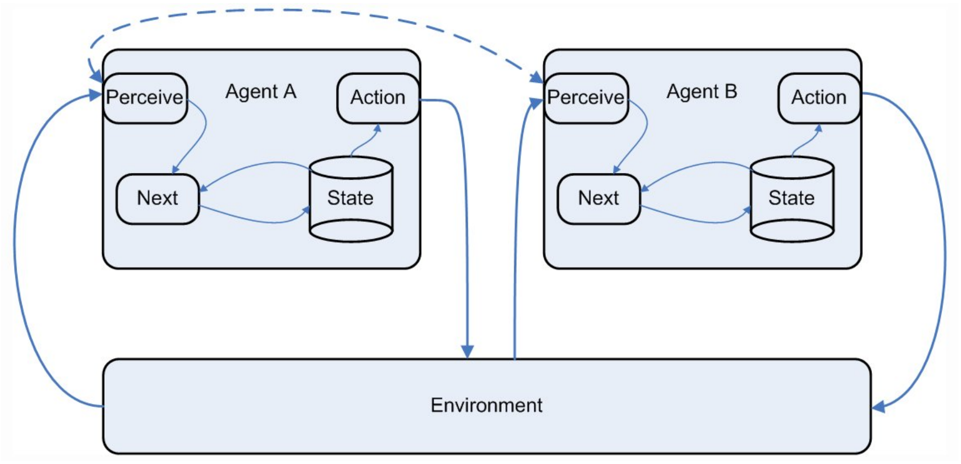


Figure 5.4: Social agents, which has the ability to interact.

5.1.4 Cognitive Agents

Cognitive agents has the ability to reason. The agent often requires an explicit representation of itself and the agents with which it interacts, which it stores in its knowledge base, and it uses this information to act [32]. The difference is that a cognitive agent has the ability to learn from its experience and the experience of others, and can act even in situations, which are not contained in its knowledge base. A cognitive agent also has the ability to reason about the outcome resulting from its actions in the environment, thus giving it planning capabilities.

5.2 The Agent Environment

The environment from the partiel subjective view of an agent is considered to be everything else than the agent itself, which the agent can perceive. In almost any realistic environment uncertanty will inherently be present [52].

An environment, in which the agent can obtain complete, accurate and up-to-date information, is referred to as accessible. Real-world environments are not accessible in that sence, thus also referred to as inaccesible.

If every action in an environment has a garanteed effect, meaning that there are no uncertanties regarding the resulting state, then we could call the environment deterministic, but in the real world environments are typically non-deterministic or considered to be.

An environment that can be assumed to remain unchanged, except when influenced of the agents actions, is considered a static environment. The real world and comprehensive problem domains are dynamic environments. When an agent in a dynamic environment performs no external action to effect the environment between time t_0 and t_1, then the agent can not assume that the environment at t_1 is the same as it was in t_0. In a static environment the agent only needs to perform information gathering once, from where it can accurately predict the effects of its actions on the environment, assuming that it understands its environment correctly and the effect of its actions.

If an environment has a finite number of actions and perceptions in it, it is considered to be discrete, else the environment is continuous.

To summarize we can categorize the environment into the following:

1. **accessible** vs. **inaccesible**
2. **deterministic** vs. **non-deterministic**
3. **static** vs. **dynamic**
4. **discrete** vs. **continuous**

5.3 What is a Multi-Agent System?

Systems that contain multiple agents are referred to as multi-agent systems a.k.a multiple agent systems [4]. Where we earliere talked about an agent in

[4]http://en.wikipedia.org/wiki/Software_agent

its environment, when introducing multiple agents, we will have to consider organizing the agents, and determine how the agents should interact.

In an organized multi-agent system, the agents characteristically have different capabilities and will not have global access to data in the environment, so in order to solve a common objective, they must interact. In a multi-agent system, data is generally decentralized and execution is asynchronous, there may be little or no global control, which is why such systems are sometimes referred to as swarm systems [52, 32].

To advance individual goals and the overall system in which agents reside, agents can communicate, coordinate and negotiate among each other.

In the following sections we will describe the widely used AEIO paradigm and BDI agent architecture.

5.4 The AEIO paradigm

A multi-agent system can be decomposed into four main entities: Agent, Environment, Interaction and Organization, which is referred to as the AEIO paradigm [32], which contains the following three statements, from which we will describe the declarative principle.

The Declarative Principle
$\mathsf{MAS} = \mathsf{A} + \mathsf{E} + \mathsf{I} + \mathsf{O}$

The Functional Principle
$\mathsf{Function(MAS)} = \sum \mathsf{Function(entities)} + \mathsf{Emergence\ Function}$

The Recursive Principle
$\mathsf{Entity} = \mathsf{basic\ entity} \mid \mathsf{MAS}$

The declarative principle is composed of the following four components:

1. **Agent:** The agent is the basic component in a multi-agent system, which we described in section 5.1.

2. **Environment:** The environment, which was described in section 5.2, is where agents evolve, and can be either virtually or physically modeled according to whether one chooses software or hardware agents.

3. **Interaction:** Interaction among the agents, can vary from simple ones like forces exerted between agents, to very complex ones like speech acts or interaction protocols. The most used interaction protocols in present MAS frameworks are described in section 5.7.

4. **Organization:** The organization of agents is the last component, which is often identified as the entire multi-agent system or the society of agents, but may also be exhibited as an independent feature of a MAS. Organization is described in section 5.6.

5.5 BDI Agent Architecture

One of the major cognitive agent class of architectures is the belief-desire-intention (BDI) agent architecture [18]. A representative BDI architecture is shown in figure 5.5.

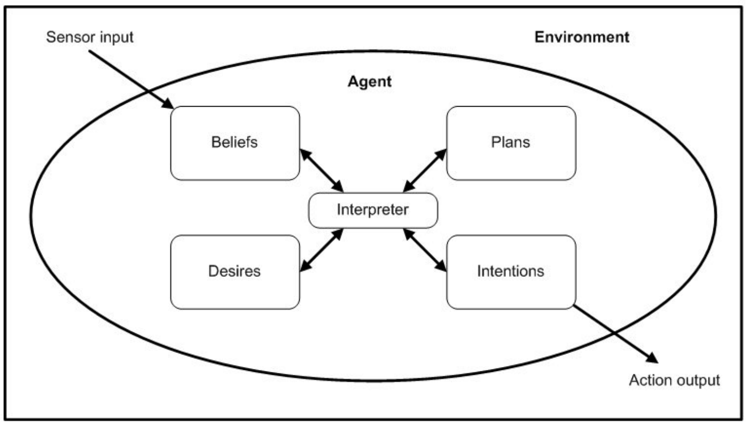


Figure 5.5: MAS BDI architecture

As it is seen from figure 5.5, the BDI architecture contains four key data structures[51], namely.

- **Beliefs:** Is the information that the agent has about the environment, which may be incomplete or incorrect. This information can be represented by variables or more generally symbolically.

- **Desires:** Is the set of tasks that the agent is able to achieve.

- **Intentions:** Is a subset of the desires, which the agent has committed to achieve. The agent will try to achieve an intention until it believes that the intention is satisfied or until the intention is no longer achievable.

- **Plans:** Is the *plan library* for the BDI agent. The *plan library* specifies the courses of actions that may be followed by an agent in order to achieve its intentions.

The interpreter shown in figure 5.5 has the following responsibility:

1. Update the agents beliefs, according to new perceptions of the environment.
2. Generate new desires on the basis of new beliefs.
3. Select a subset of the tasks from the desires and let these tasks become the intentions of the agent.
4. Select a course of action, based on the intentions and plan library.

Rao and Georgeff has defined an abstract interpreter for the BDI architecture[43]. This interpreter is shown below

BDI-interpreter

```
1  initialize-state();
2  repeat
3    options := option-generator(event-queue);
4    selected-options := deliberate(options);
5    update-intentions(selected-options);
6    execute();
7    get-new-external-events();
8    drop-succesful-attitudes();
9    drop-impossible-attitudes();
10 end repeat
```

We will now give a brief description of this interpreter. Line 1 initializes the state of the agent, that is the information, motivational and deliberative states of the agent (These states represents the *mental attitudes* of Belief, Desire and Intention respectively[43]). Then an execution cycle is entered wherein at line 3 the desires of the agent are calculated (The event-queue contains the perceptions of the environment). Line 4 and 5 updates the intentions with a subset from the desires. Line 6 executes any intention if it is possible according to the *plan library*. Line 7 adds perceptions, that have occured during the interpreter cycle, into the event-queue. In line 8 and 9 the structures of desires and intentions are modified, such that all succesful desires and satisfied intentions are dropped, as well as impossible desires and unrealisable intentions. The execution cycle is then repeated.

5.6 Organization of Agents

The collection of all agents in a multi-agent system, is referred to as a society of agents. When designing a multi-agent system, one must consider how to structure the society of agents, which is known as organizing the agents [32]. The term organization refers to placing agents that has a certain unity into groups. Groups, that can be represented explicitly, are furhtermore referred to as organizations.

The following description of three types of hierarchical architectures have been adapted from [32]:

1. **Simple hierarchy:** The simple hierarchy consists of only two levels in a tree, where the top level, is the controlling part of the hierarchy, that makes the decisions, and the bottom level nodes each solve a subproblem of the overall problem. This hierarchy can be viewed as having a master-slave relationship between the two levels.

2. **Multi-level hierarchy:** A multi-level hierarchy consists of multiple complementary levels, and thereby allowing control to appear at various levels. Nodes within the same level have the ability to interact, and thereby coordinate their effort through negotiation. The overall diffence from the simple hierarchy is that no node has complete control of its sublevel, since all nodes within a level can communicate.

3. **Decentralized hierarchy:** The decentralized hierarchy is a multi-level hierarchy with the difference, that there are no interaction between nodes in a level, and every node can be an organization itself. The advantage of this hierarchy is that the control is distributed, and each level chooses its own organization, thereby giving it more control capabilities, making decisions decentralized. The higher the level, the more power regarding long term operations, thus the top level has the overall decision power regarding the long term operations in the entire society.

5.7 Agent Interaction

Interaction is one of the components in the AEIO paradigm. It is a key feature in MAS, because agents needs to interact to solve large and complex problems. Figure 5.6 combined from [52, 34] shows how agents can be organized in a MAS and their structure for interaction.

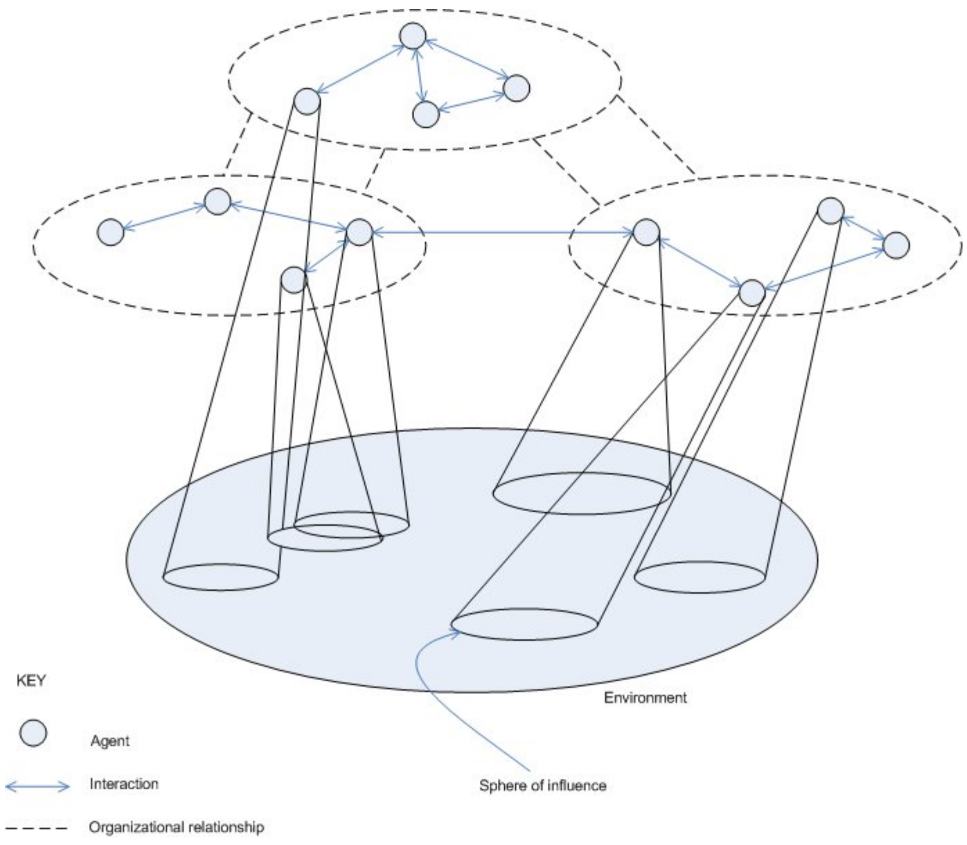


Figure 5.6:

Interaction in a Multi-Agent System has the following meaning[5]:

> *Interaction is a kind of action that occurs as two or more objects have an effect upon one another.*

Interaction ranges from simple forces between agents to complex ones like speech acts or interaction protocols[32]. Another interaction kind is the blackboard interaction model, which for example is used by the Cougaar agent framework(see section 12 for further information about Cougaar).

Speech act theory is the foundation for agent Interaction Languages. A speech act contains three acts[18]:

- **Locutionary Act:** uttering words and sentences with meaning
- **Illocutionary Act:** type of action, *intent* of the utterance

[5]http://en.wikipedia.org/wiki/Interaction

- **Perlocutionary Act:** expected *(desired) result* of the utterance

Example of a speech act from [8]:

> When a person utteres "Shut the door!" to another person, we have the following three acts:
>
> *The Locutionary act* is the physical utterance with context and reference, i.e. who is the speaker and the hearer, which door, etc.
>
> *The Illocutionary act* is the intention of the utterance, i.e. the speaker wants the hearer to close the door.
>
> *The Perlocutionary act* occurs as a result of the illocution, i.e. the hearer closes the door.

Interaction Languages[6] is used by agents to communicate. The most known and widely used ones are KQML and FIPA-ACL. We will now describe these two ACL's.

5.7.1 KQML

KQML is an agent communication language initially defined by *"The DARPA Knowledge Sharing Initiative External Interfaces Working Group"* and is described in [21]. The group was formed in 1990 to develop protocols that could be used by autonomous information systems to share knowledge. In 1997 Yannis Labrou and Tim Finin proposed a new specification for the KQML [36], where there are significant changes to reserved performatives, their meaning and their use. Some of the KQML reserved performatives and their description can be seen in table 5.1.

A KQML message is composed of three parts:

1. Performative 2. Parameter 3. Parameter Value

An example of an "ask-if" KQML message and the three parts can be seen in figure 5.7. In this example the D-Planner Agent asks the Kamag4843 Agent whether it's state is free.
When following reserved parameters (shown in table 5.2) are used in a KQML message, then they have to conform with the meanings in this table.

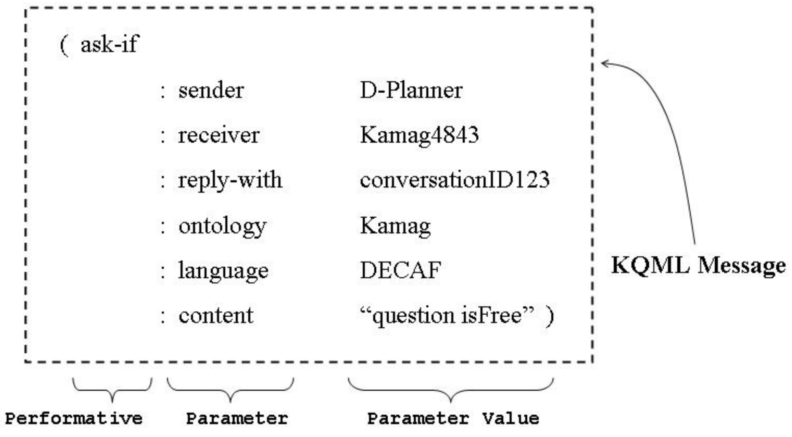
[6]Also called Agent Communication Languages (ACL's)

Name	Meaning
ask-if	S wants to know if the :content is in R's VKB
ask-all	S wants all of R's instantiations of the :content that are true of R
ask-one	S wants one of R's instantiations of the :content that is true of R
tell	the sentence is in S's VKB
untell	the sentence is not in S's VKB
deny	the negation of the sentence is in S's VKB
insert	S asks R to add the :content to its VKB
uninsert	S wants R to reverse the act of a previous insert

Table 5.1: Some KQML reserved performatives

Parameter	Meaning
:sender	the actual sender of the performative
:receiver	the actual receiver of the performative
:from	the origin of the performative in :content when forward is used
:to	the final destination of the performative in :content when forward is used
:in-reply-to	the expected label in a response to a previous message (same as the :reply-with value of the previous message)
:reply-with	the expected label in a response to the current message
:language	the name of the representation language of the :content
:ontology	the name of the ontology (e.g., set of term definitions) assumed in the :content parameter
:content	the information about which the performative expresses an attitude

Table 5.2: KQML reserved parameters



```
( ask-if
        : sender        D-Planner
        : receiver      Kamag4843
        : reply-with    conversationID123
        : ontology      Kamag
        : language      DECAF
        : content       "question isFree" )
```

Figure 5.7: A KQML message

Name	Meaning
Accept Proposal	The action of accepting a previously submitted proposal to perform an action.
Call for Proposal	The action of calling for proposals to perform a given action.
Failure	The action of telling another agent that an action was attempted but the attempt failed.
Inform	The sender informs the receiver that a given proposition is true.
Propose	The action of submitting a proposal to perform a certain action, given certain preconditions.

Table 5.3: Some FIPA-ACL performatives

5.7.2 FIPA-ACL

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems [1].

The FIPA-ACL is similar to KQML and it also defines performatives[7]. A performative in FIPA is called a communicative act. Some of these performatives and their meaning can be seen in table 5.3.

An example of a FIPA-ACL message can be seen in figure 5.8 [52]. An advantage of the FIPA-ACL is that every communicative act are described by

[7]The defined performatives are listed in the FIPA Communicative Act Library Specification [22]

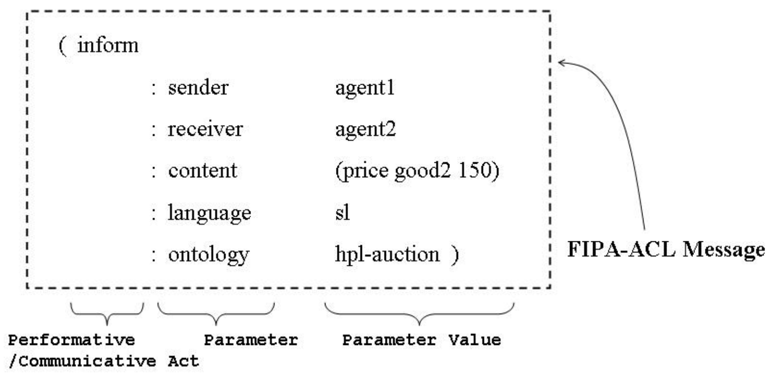


Figure 5.8: A FIPA-ACL message

a Semantic Language (SL). SL is a quantified, multimodal logic with modal operators for beliefs (B), desires (D), uncertain beliefs (U), and intentions (persistent goals, PG) [35]. The communicative act inform has the following SL:

$$\begin{aligned}&\langle i, inform(j, \varphi)\rangle\\ &FP : B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)\\ &RE : B_i\varphi\end{aligned}$$

The first part "$\langle i, inform(j, \varphi)\rangle$" says that agent *i* informs agent *j* of content φ.

The second part "$FP : B_i\varphi \wedge \neg B_i(Bif_j\varphi \vee Uif_j\varphi)$", defines the Feasibility Precondition (FP), which is the necessary condition which has to be fulfilled before agent *i* can use the communicative act inform. This FP says that before agent *i* can inform agent *j* of content φ, agent *i* has to Believe φ and Not Believe that agent *j* Believes anything about φ or are Uncertain about φ. That is agent *i* believes φ and believes that agent *j* doesnt know anything about φ.

The last part "$RE : B_i\varphi$", defines the Rational Effect (RE), which is the effect, agent *i* expects, that will occur when it uses communicative act inform. It says that the result of the communicative act will be to make agent *j* Believe φ. But it is a rational effect, that is even if agent *i* believes that the outcome of the communicative act will be to make agent *j* believe φ, then there is no guarantee that it will happen; it is only what agent *i* believes will happen.

Another advantage that FIPA-ACL has is that there have been specified specific Interaction Protocols (IP) for different purposes. These IP's can be

Interaction Protocol
Request Interaction Protocol
Query Interaction Protocol
Request When Interaction Protocol
Contract Net Interaction Protocol
Iterated Contract Net Interaction Protocol
English Auction Interaction Protocol
Dutch Auction Interaction Protocol
Brokering Interaction Protocol
Recruiting Interaction Protocol
Subscribe Interaction Protocol
Propose Interaction Protocol

Table 5.4: FIPA Interaction Protocols

seen in table 5.4.

Among the protocols in table 5.4 the Contract Net Interaction Protocol specifies the interaction between agents for fully automated competitive negotiation through the use of contracts [7]. Figure 5.9 shows the UML diagram for this interaction protocol. This protocol is used whenever one agent(the Initiator) takes the role as a manager, wanting one or more agents(Participants) to perform a certain task. For a given task, any number of the Participants may respond, where some gives a proposal and some refuses; negotiations then continue with the Participants that proposed [23].

Agents can also interact via blackboards instead of communication with ACL's. In the next subsection we describe the publish-subscribe model which is used by the Cougaar agent framework.

5.7.3 Publish-Subscribe model

In a publish-subscribe model a sender will publish a message, without specifying a receiver. A subscriber subscribes to a specific kind of a message, and whenever someone publishes that kind of a message, the subscriber will be notified and receives the message. There exists two categories of publishing/subscribing as follows[8]:

- Broker: There exists a broker agent, which manages all subscriptions. Whenever an agent publishes a message, the broker checks the subscriptions and forwards the message to the agents which made a subcription for that kind of a message.

[8]http://en.wikipedia.org/wiki/Publish/subscribe

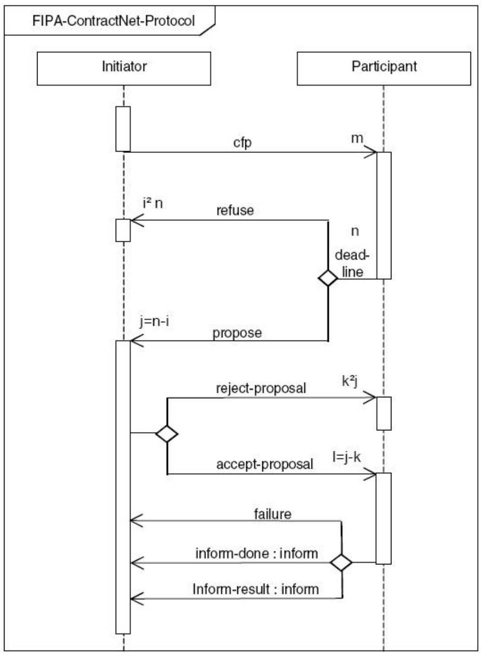


Figure 5.9: FIPA Contract Net Interaction Protocol

- Non-broker: Every agent manages individually the subscriptions from other agents, and when the agent publishes a message, all subscribed agents will receive it automatically.

The Cougaar framework is a non-broker publish-subscribe model. Each agent in Cougaar has a local blackboard, where it can publish objects and other agents can subscribe to those objects. The subscribing agents will then be notified whenever an object in the blackboard is created, modified or removed (see appendix E for a description of the Cougaar framework).

5.8 Framework requirements

The problem with MAS is that there are no official standard of how to design such a system, thus various multi-agent frameworks have been developed for specialized problem domains in the past decade.

In the research of the most suitable MAS frameworks, we have gone through 128 MAS frameworks[9]. Prior to the investigation, we felt the need to compose an overall list of requirements to a suitable framework in order to

[9]http://eprints.agentlink.org/view/type/software.html

narrow down the search.

We have decided that the MAS framework we will work with, should have the following features:

1. Surviveability
2. Java Toolkit
3. Planning module
4. GPGP
5. Agent communication language **ACL**(FIPA/KQML)
6. Open source
7. Succes - has it been used in other projects before
8. Well documented API
9. Well documented
10. BDI

5.9 Suitable Frameworks

In this section we present the ten most suitable MAS frameworks in respect to our thesis project based on our framework requirements(see section 5.8). The following ten frameworks presented in this section are all candidates for being used in our thesis project. In the elimation process we weighted that the frameworks should not be too similar. We end this chapter by a summary that lists the framework features, which are based on our requirements.

5.9.1 A-globe

A-globe is a fast and lightweight platform with agent mobility support and communication inaccessibility [48]. The system architecture for A-globe can be seen in figure 5.10.

An A-globe platform[10] runs in its own JVM instance and consists of one or more Agent Containers.

Agents live in those containers and are able to communicate (XML or binary) and migrate to other containers. If an agent uses a library not defined

[10]Its possible to run a maximum of 1000 platforms on a single machine

in the platform to which it is migrating to, then the library manager moves this library to the new platform.

The extended version includes Geographical Information System (GIS), which maintains information about agent visibility. The Environment Simulator (ES) agent sets this visibility, and two agents can communicate, only if they are visible to each other.

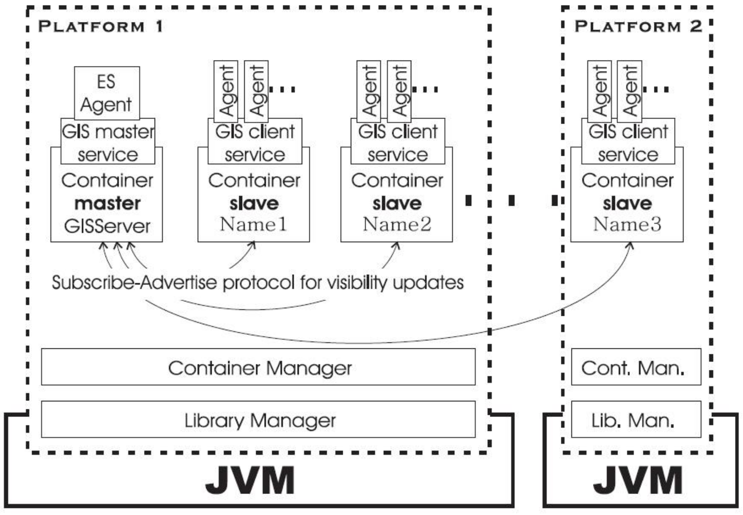


Figure 5.10: A-globe system architecture

5.9.2 AgentBuilder

The commercial framework AgentBuilder is an integrated tool for constructing intelligent software agents[11]. Agent Builder is an extension of the Shoham's AGENT-0 and Thomas's PLAnning Communicating Agents (PLACA), both are agent programming languages [46, 33]. The agents have Mental Models which consist of Beliefs, Commitments, Intentions and Capabilities. The agent architecture in AgentBuilder can be seen in figure 5.11.

The AgentBuilder consists of a Toolkit and a Runtime-system. The toolkit helps the agent developer to analyze and design the multi-agent system. It offers the developer a couple of tools so that constructing a multi-agent system can be done fairly quickly, e.g. the Agent Manager tool is used to define the agents initial- beliefs, commitments, intentions etc. Another tool is the Agency Manager, which is used to form groups of agents which can

[11]http://www.agentbuilder.com/Documentation/product.html

communicate and cooperate to solve some tasks.

The Runtime-system runs in a JVM and consists of an Agent Engine, where agents are executed. The problem-specific code that each agent requires for its operation can be written in Java, C or C++. This code are placed in entities which is called Project Accessory Classes (PACs).

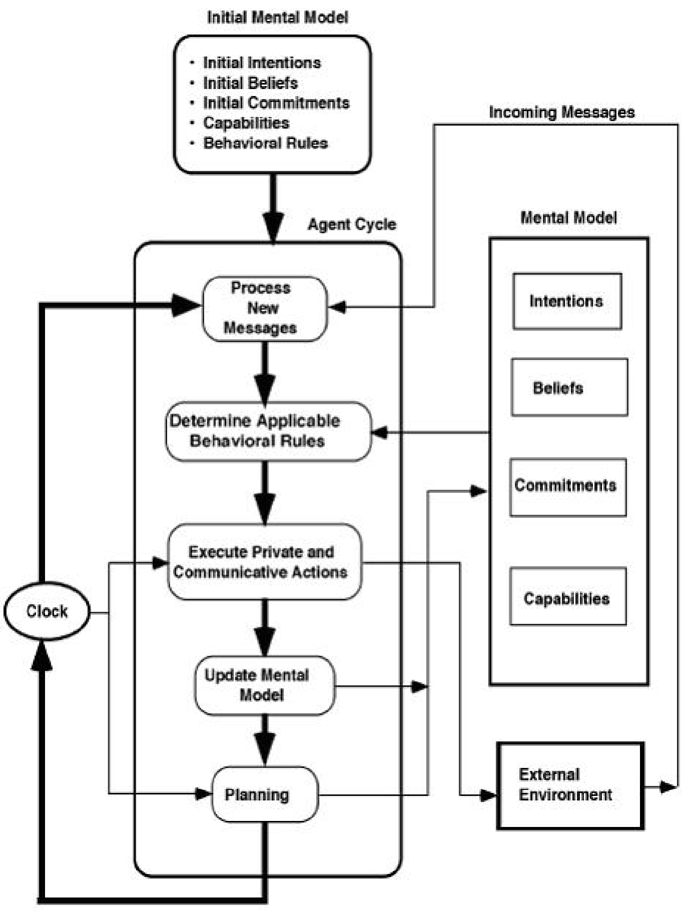


Figure 5.11: AgentBuilder agent architecture

5.9.3 Cougaar

The Cognitive Agent Architecture (Cougaar) is an open source project, resulted from two consecutive DARPA research projects spanning over multiple years[12]. Cougaar is a Java-based architecture designed for construct-

[12]http://eprints.agentlink.org/5343/

ing large-scale distributed Multi-Agent Systems. Along with the architecture Cougaar provides demonstration, visualization and management components, which will simplify the development of complex distributed applications[13], and all the source code can be downloaded from the Cougaar's open source project site www.cougaar.org.

Cougaar has shown to be very efficient and has demonstrated that using advanced agent-based technology it is feasible to conduct rapid, large scale distributed logistics planning and replanning. The Cougaar project has also been focused on survivability of the distributed agents-based systems, specifically for systems operating in extremely chaotic environments [20], meaning more reliable and dynamic systems, which are more robust for changes in the physically environment.

Since the end of 2004 Cougaar has not been DARPA-sponsored, but periodic stable releases of the Core Architecture are still provided by the community[14].

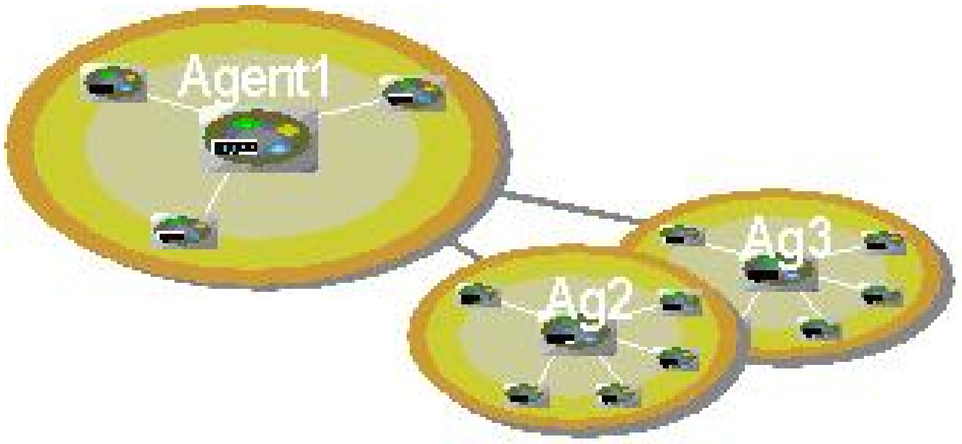


Figure 5.12: Cougaar Architecture

5.9.4 DECAF

DECAF[15] is a toolkit for developing Multi-Agent Systems. DECAF is a kind of a BDI-model [27]. An agent in this framework has some capabilities which are called "desires" in BDI. The interesting thing is that the BDI

[13]http://en.wikipedia.org/wiki/Cougaar
[14]http://cougaar.cougaar.org/software/latest/doc/roadmap.html
[15]Distributed, Environment-Centered Agent Framework

“intentions” in this framework are partitioned into 3 levels namely “planning”, “scheduling” and “executing”. The architecture for an agent in DECAF can be seen in figure 5.13.

The toolkit runs in a JVM, and has additional tools to help the developer implementing a Multi-Agent System. One of the tools is the PlanEditor, which is a graphical interface [42]. A developer can easily with the PlanEditor, define what agents are able to do, that means which actions that they are able to take.

As seen from figure 5.13 every agent uses a plan-file. The plans and their execution structure in DECAF is an extension of RETSINA and TAEMS, but the complexity is less than TAEMS [25].

There is also a GPGP module in DECAF. The basic idea with GPGP is that each agent constructs its local view of the structure and relationships of its intended tasks [26].

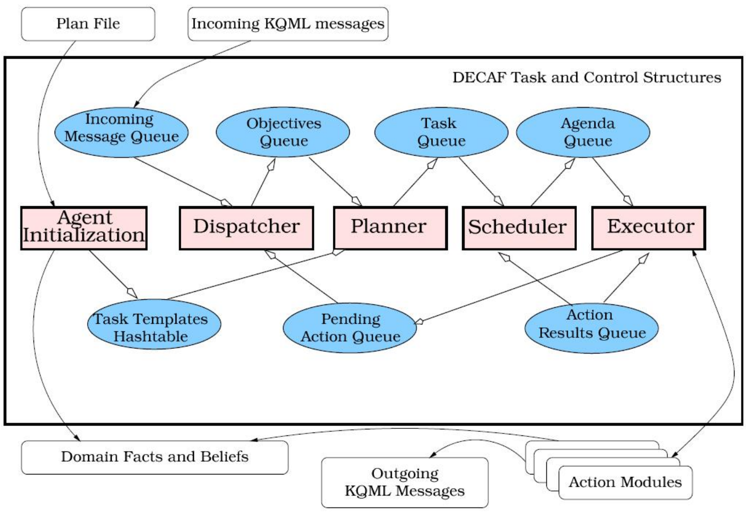


Figure 5.13: Decaf architecture. Agents communicate with KQML messages.

5.9.5 JACK

JACK is a Java-based Multi-Agent framework. It has a lightweight implementation of the BDI architecture and supports extension by other agent models. The two systems, Procedural Reasoning System (PRS) and dMARS, are both predecessors to JACK, and both implemented the BDI model [40].

JACK is not bound to a specific agent communication language such as KQML or FIPA ACL, but has the capability to use them if needed. It provides a native lightweight communications infrastructure for situations where high performance is required.

JACK supports creation of teams and roles for agents by the means of an extension called *SimpleTeam*. Agents in JACK are social agents, that is both proactive(goal directed) and reactive(event driven) [41].

The JACK framework consists of several components, which are listed beneath [40]:

- **JACK runtime environment**[16]: Agents are executed here. Communication and concurrency issues with a simple agent naming service are also handled here.
- **JACK compiler**: Compiles JACK Agent Language into Java code.
- **BDI Component**: Adds support for BDI reasoning.
- **SimpleTeam Component**: Adds support for team-based reasoning.
- **Agent Development Environment**: is a GUI for viewing and manipulating JACK applications.
- **Agent Debugging Environment**: is for viewing messaging between agents and for displaying internal execution states in the kernel.
- **JACOB**: is a toolkit supporting conversion of messages and objects into human readable textual format, fast binary format or XML.

JACK is built by the company "Agent Oriented Software" (AOS), and is not freeware.

5.9.6 JADE

JADE[17] is a Multi-Agent Framework developed by TILAB [10]. JADE is used to develop distributed multi-agent applications. JADE includes following [47]:

- **A runtime environment** where agents can live and execute.
- **A library** of classes(Java) to develop agents.

[16]Also called the JACK agent kernel

[17]Java Agent DEvelopment Framework

- **Graphical tools** that allows administrating and monitoring the activity of running agents.

A runtime environment contains one "Container" where the agents reside. A standalone Container or several linked Containers are called a "Platform", se figure 5.14. Containers can be main- or "slave" containers, the "slave" Containers have to register with the main Container [47].

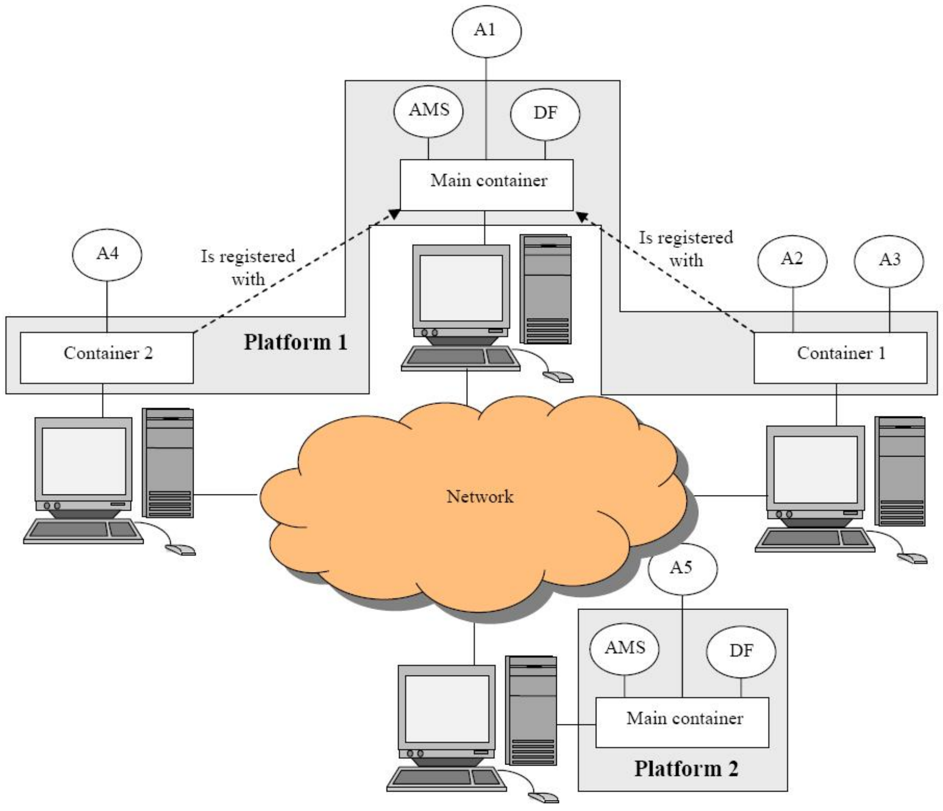


Figure 5.14: JADE Containers and Platforms

As seen from the figure each Main Container contains an AMS[18] and a DF[19] agent. The AMS ensures that each agent in the platform has a unique name and can create/kill agents in the platform if requested. The DF provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

[18]Agent Management System
[19]Directory Facilitator

JADE is compliant with the FIPA specifications, and JADE agents can therefore interoperate with other agents, provided they also comply to the FIPA specifications.

JADE does not deal with the internal of agents[20], that is JADE agents are not intelligent agents. JADE only deals with aspects external to the agent, such as message transport, agent lookup, agent life-cycle etc. The life cycle of an JADE agent which is FIPA compliant are shown in figure 5.15.

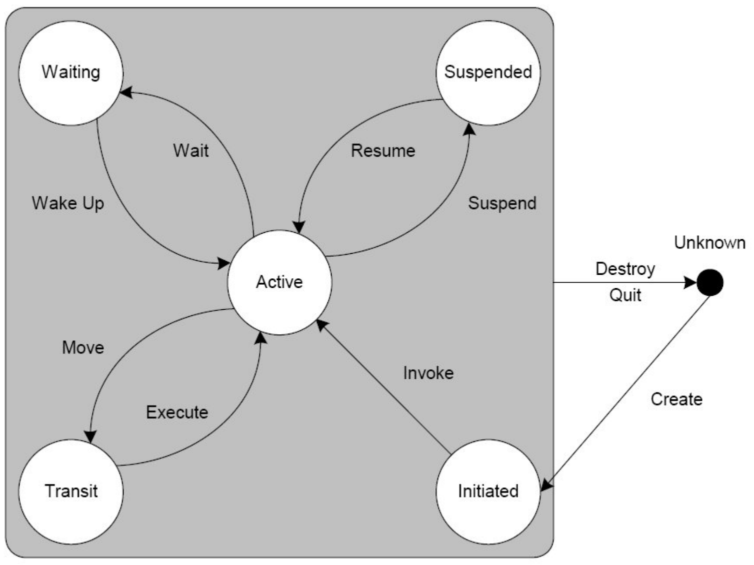


Figure 5.15: JADE agent life cycle

As seen from the figure a JADE agent can be in one of several states[24], which are described below:

- **Initiated:** the Agent object is built, but hasn't registered itself yet with the AMS, has neither a name nor an address and cannot communicate with other agents.
- **Active:** the Agent object is registered with the AMS, has a regular name and address and can access all the various JADE features.
- **Suspended:** the Agent object is currently stopped. Its internal thread is suspended and no agent behaviour is being executed.
- **Waiting:** the Agent object is blocked, waiting for something. Its internal thread is sleeping on a Java monitor and will wake up when some condition is met (typically when a message arrives).

[20]http://jade.tilab.com/description-quickinfo.htm

- **Deleted:** the Agent is definitely dead. The internal thread has terminated its execution and the Agent is no more registered with the AMS.
- **Transit:** a mobile agent enters this state while it is migrating to the new location. The system continues to buffer messages that will then be sent to its new location.

5.9.7 Jason

Jason is a Java-based interpreter for an extended version of AgentSpeak[21].

Besides interpreting the original AgentSpeak(L) language, Jason also features:

- strong negation, so both closed-world assumption and open-world are available;
- handling of plan failures;
- speech-act based inter-agent communication (and belief annotations on information sources);
- annotations on plan labels, which can be used by elaborate (e.g., decision theoretic) selection functions;
- support for developing Environments (which are not normally to be programmed in AgentSpeak; in this case they are programmed in Java);
- the possibility to run a multi-agent system distributed over a network (using SACI);
- fully customisable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting);
- a library of essential internal actions
- straightforward extensibility by user-defined internal actions, which are programmed in Java.

Jason is a Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net. AgentSpeak(L) has been one of the most influential abstract languages based on the Belief Desire Intention (BDI) architecture. The type of agents specified with AgentSpeak(L) are sometimes referred to as reactive planning systems. To the best of our knowledge, Jason is the first fully-fledged interpreter for a much improved version of

[21]http://jason.sourceforge.net/

AgentSpeak, including also speech-act based inter-agent communication [11].

5.9.8 MadKit

MadKit is a multi-purpose, distributed multi-agent platform used in various project over the entire world. Especicially the MadKit framework is used for researching purposes, since this platform is open source and developed with Java and free for download an usage. Developers can use Madkit to make basic products used in commercial applications[22].

The concept of MadKit is based on micro-kernel and agent-identification services. The micro-kernel is small, thus the agents offer the most important services needed in ones application.

The MadKit framework is organizational oriented, and uses agents, roles, groups as standard components for building complex applications based on MAS. There are no requirements for the internal structure of an agent, thus developers can customize agents to satisfy their needs. Communication in MadKit is socket-based, allowing decentralized agent architectures.

MadKit provides a graphical environment for development of agent systems, as well as a graphical runtime agent environment. Using this facility, developers can create simple environments and agents, without having to think about designing a graphical environment, since it is provided.

Within minutes one can download and install the MadKit framework from their website[23], and it is highly usable for developing fast and simple agent based applications.

5.9.9 MPA

The **M**ulti-Agent **P**lanning **A**rchitecture (**MPA**) is a research program within the **D**ARPA/**R**OME **P**lanning **I**nitiative (**ARPI**)[24] which focuses on studies and academic research into the area of military operational planning and scheduling in order to address tomorrow's technology.

MPA is organized around the concept of a planning cell, wherein a collection of agents is committed to one particular planning process[50]. A planning cell contains a planning cell manager agent and plan server agent. The planning cell manager agent has a planning cell from a community

[22]http://www.madkit.net/site/madkit/doc/userguide/userguide.html
[23]http://www.madkit.net
[24]http://www.ai.sri.com/ wilkins/mpa/

of agents, and distributes planning task among selected agents. The central repository for plans is the plan server agent, which holds plan-related information during the course of planning a task, that is obtained from accepting incoming information from Planning Agents(PA), then by performing processing, storing information, and making this information available to any PA through queries. Agents communicate with KQML messages.

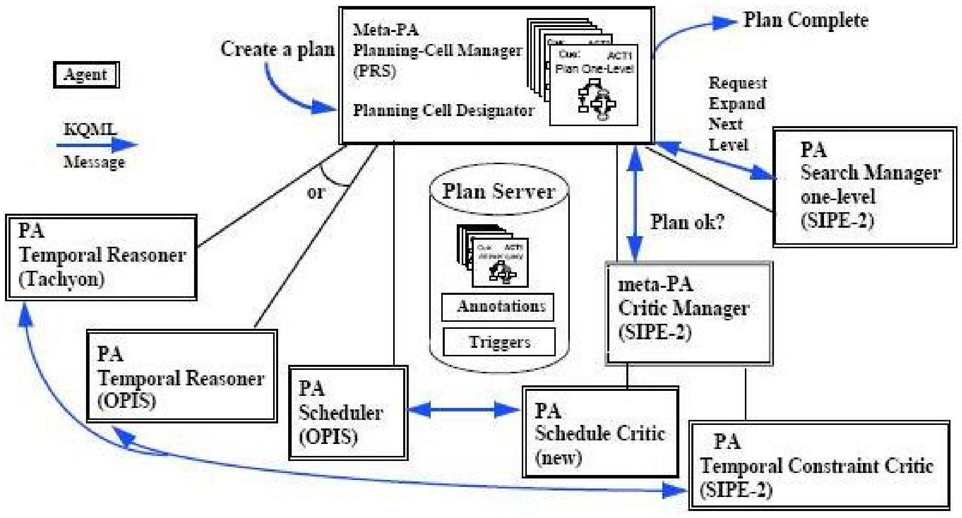


Figure 5.16: MPA single planning cell

MPA concerns Multi-Agent Planning, visualization and simulation, which specifically addresses a human planner's ability to rapidly obtain multiple, significantly different alternative courses of action, evaluate them, select a primary candidate, and have that primary plan fleshed-out and tuned to take into account special requirements and considerations. Part of the challenge is balancing the need to have humans provide ultimate control and oversight vs. the need to respond with plans and schedules very quickly.

MPA provides services to a range of requests by defining a range of generic planning agents. Agents in MPA has various properties, such as providing partial or overall plans, reporting incremental progress, and by continuously responding to new conditions, constraints and suggestions. It is the meta-Planning Agents, which holds specialized knowledge about strategies for dividing work, conflict resolution and future plan merging. Every meta-Planning Agent has a collection of PA's and other planning clusters. A meta-PA is responsible for coordinating the activities of the collection of PAs and other planning clusters.

The MPA framework has proven its usability to large-scale problem solving, i.e. the Air Campaign Plannning (ACP), which integrated a set of technologies such as scheduling, temporal reasoning simulation and visualization. Development and evaluation of a complex plan with over 4000 nodes has cooperated these technologies [49]. Demonstrations shows multiple asynchronous agents cooperatively generating a plan or set of alternative plans in parallel, a meta-PA reconfiguring the planning cell during planning, and agents running on different machines both locally and over the Internet. MPA demonstrations employ technologies developed outside SRI, show a flexible and novel combination of planning and scheduling techniques, and demonstrate dynamic strategy adaptation in response to partial results.

5.9.10 ZEUS

Many toolkits and frameworks have been developed over the past decade to aid agent development. ZEUS provides class libaries and user-customizable components for users of the Java programming language, since the motivation for building the ZEUS agent building toolkit, was to create a general framework based on general architectures and methologies, for developing collaborative agent systems, wherein agents work together to archieve a shared goal [38].

In order for the agents to communicate a common transport protocol is required, and an inter-agent communication protocol. Agents must be able to reason, in order to know when to reqeust and release resources, and when to collaborate with some other agent, in order to fulfil its goal.

We can categorise the ZEUS toolkit into three main functional groups: an agent component libary, a visualization tool, and agent building software(see figure 5.17):

1. **The agent component libary:** The first functional group is the agent component libary, wherein java classes form the building bloks of the agents. All classes are composed in a Java package for easy usage. Using this package all the functionallity required in order to implementing collaborative agents are in place. For communication purpose among agents a performative-based inter-agent communication language called KQML is included in the toolkit, which is expected to be upgraded to FIPA´s ACL protocol for inter-agent communication in the future, thus the message-passing system is currently asynchrounous socket-based. A generic planning and scheduling system is also included in ZEUS which makes it useful to our project.

2. **The visualization tool:** The second functional component gives the

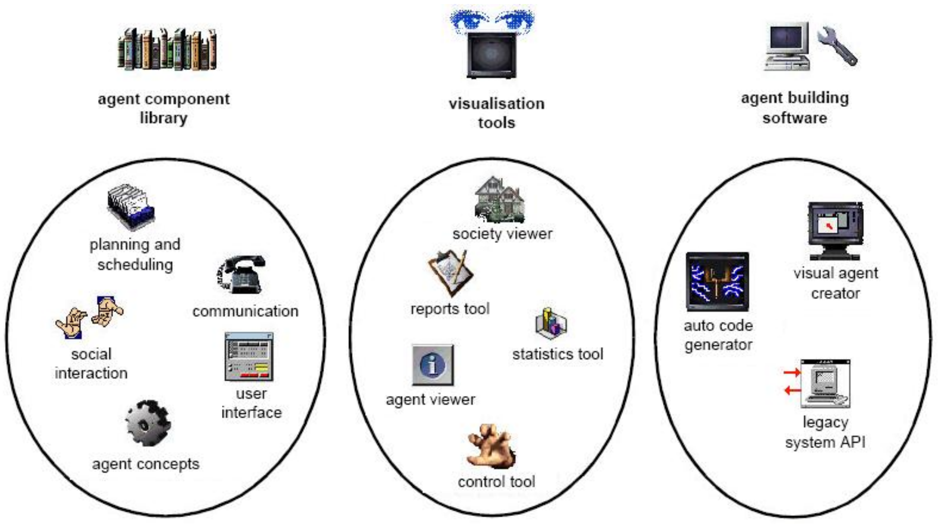


Figure 5.17: ZEUS: functional groups of the class libary

developer a posibility of monitoring the agents, one can look at theese visualization tools as a place to collect global information about all agents, which is quite good for debugging purpose and for making statistics.

3. **The agent building software:** Application programmers are allowed to monitor changes in the internal state of an agent, by using an event model along with an API.

Figure 5.18 which have been adapted from [37], shows the architecture of a generic ZEUS agent. We briefly describe the different elements in figure 5.18:

- Mailbox: handles communication between agents. A complex entity, containing a server that accepts incoming messages.
- Message handler: processes incoming messages and dispatches them to components within the agent.
- Coordination engine: coordinates the agents overall activities with other agents based on the strategies specified in its knowledge base. Is responsible for decision making, goal pursuing, goal abandoning.
- Acquaintance model: describes the agents relation to the society in which it resides, and its beliefs concerning capabilities of other agents.
- Planner and scheduler: Plans the agents tasks based in its knowledge base. Is dependent on the results generated by the coordination engine, and the available definitions of tasks and resources.

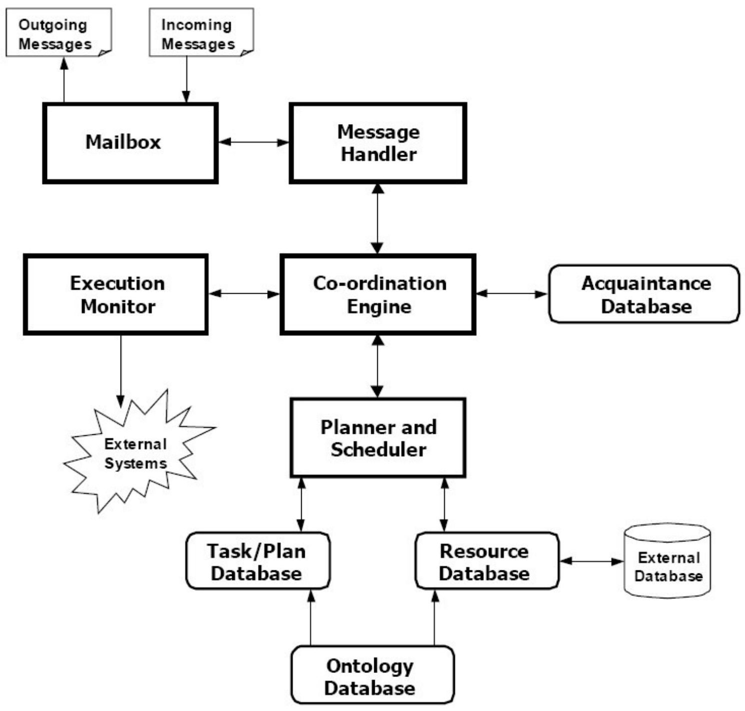


Figure 5.18: Architecture of a generic ZEUS agent

- Resource database: Lists the resources, that are available and and owned by the agent.
- Ontology database: stores the logical definition of each fact type, its legal attributes, the range of legal values for each attribute, any constraints between attribute values, and any relationships between the attributes of the fact and other facts.
- Task/Plan database: provides logical descriptions of planning operators (or tasks) known to the agent.
- Execution monitor: starts, stops and monitors external systems, which have been scheduled to run, or terminated by the planner and scheduler. It informs the coordination engine with result status.

5.10 Summary

In this chapter we have described the MAS technology including various types of agents(sections 5.1.1 - 5.1.4). Furthermore we have described the

Feature / Framework	**A-globe**	**AgentBuilder**	**Cougaar**	**DECAF**	**JACK**	**JADE**	**Jason**	**MadKit**	**MPA**	**Zeus**
Survivability	✔	✔	✔	✔	✔	✔	(✔)	✔	✔	(✘)
JAVA Toolkit	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔
Planning module	✘	✔	✔	✔	✔	✘	✔	✘	✔	✔
GPGP	✘	✘	✘	(✔)	✘	✘	✘	✘	✘	✘
ACL (FIPA/KQML)	(✔)	✔	✘	✔	(✔)	✔	✘	✔	✔	(✔)
Open Source	✔	✘	✔	✔	✘	✔	✔	✔	✘	✔
Success	✔	(✘)	✔	✔	✔	✔	(✘)	✔	(✘)	✔
Well documented API	✔	✔	✔	✘	✔	✔	✘	✔	✘	✔
Well documentation	✔	✔	✔	✔	✔	✔	(✔)	✔	✘	✔
BDI	✘	✔	✘	(✔)	✔	✘	✔	✘	(✘)	✘

Table 5.5: Features of suitable MAS frameworks

AEIO paradigm(section 5.4), which concerns composing a multi-agent system into agents, environment, interaction and organization. We have briefly described the BDI agent architecture(section 5.5).

We have described ten different MAS frameworks that are all candidates for being used in our thesis project. The frameworks and their features are summarized in table 5.5.

We have chosen to design our multi-agent system in the Decaf and Cougaar frameworks respectively. The reason for this choice have been that they both satisfy most of the requirements as seen in table 5.5. These two frameworks are very different, which have also influenced our decision. Futhermore the Cougaar framework have been used to solve logistic problems before in projects like ALP and Ultra*Log as described in section 3.2 and 3.3 respectively. Likewise Decaf contains a GPGP module for planning and coordination purposes, which have proven to be useful to solve logistic problems in other related projects, such as the hospital scheduling problem described in section 3.1. Both frameworks are open source, well documented and Java based, which we also considered to be of major importance.

✔ = supported feature, (✔) = partly supported feature
✘ = unsupported feature, (✘) = probably unsopperted feature / Unknown

Chapter 6

Middleware

In this section we are going to understand the term middleware, and why we need it. At last we then look at some various types of it.

6.1 Definition and middleware types

The following definition of middleware have been adapted from the encyclopedia site Wikipedia[1]:

> *Middleware is the enabling technology of Enterprise application integration. It describes a piece of software that connects two or more software applications so that they can exchange data.*

This definition describes middleware as software, which has the ability to connect two or more software applications, so that the applications can understand each other and exchange data.

6.2 The need for a Middleware Application

In this thesis we will design a simulation model with Automod, and two multi-agent systems(Decaf and Cougaar). The simulation model should be used for visualizing the OSS environment, including static elements like buildings, storage locations, the road system, and dynamic elements, such as the KAMAG vehicles and the ship blocks. Since the dynamic changes are controlled by the multi-agent systems, the middleware should make it possible for the simulation model and the two multi-agent systems to exchange data.

[1]http://en.wikipedia.org/wiki/Middleware

To simplify and to modularize the complete system, we have chosen to use a middleware application. The system which is going to be built can be seen in figure 6.1.

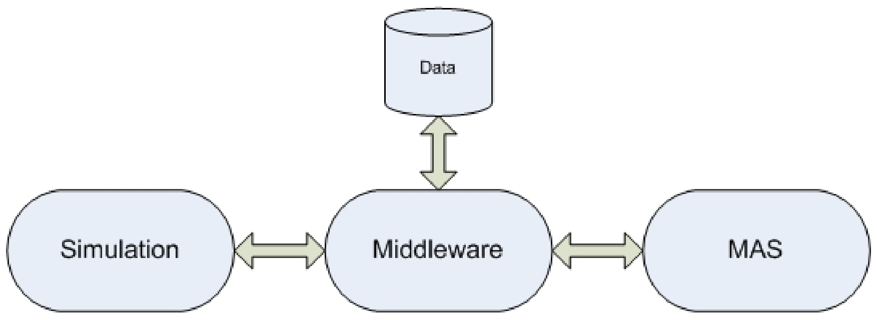


Figure 6.1: System

The advantage of using a middleware application is that we can built a part of the overall system, as shown in figure 6.2, meaning that we can built the simulation model and control it by means of a feeder, without looking at MAS at all. The feeder is just a GUI-application, where a user can enter some commands and see the reactions in the simulation model. So the user sitting in front of the feeder application is acting as an agent. This approach makes it possible to verify the simulation model before connecting it with a multi-agent system.

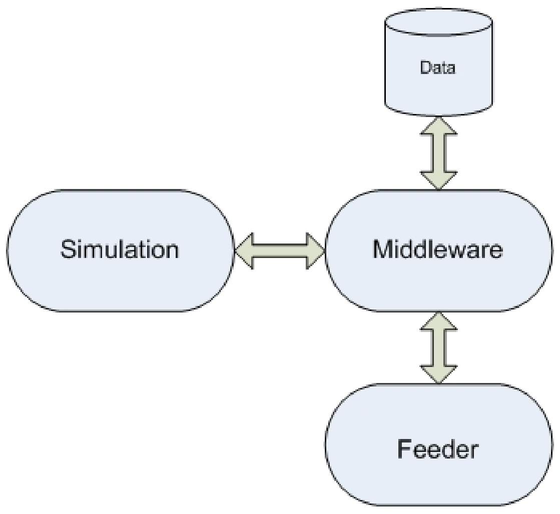


Figure 6.2: System with Feeder

When the system shown in figure 6.2 is implemented, the multi-agent system can be designed, resulting in a complete system as shown in figure 6.3.

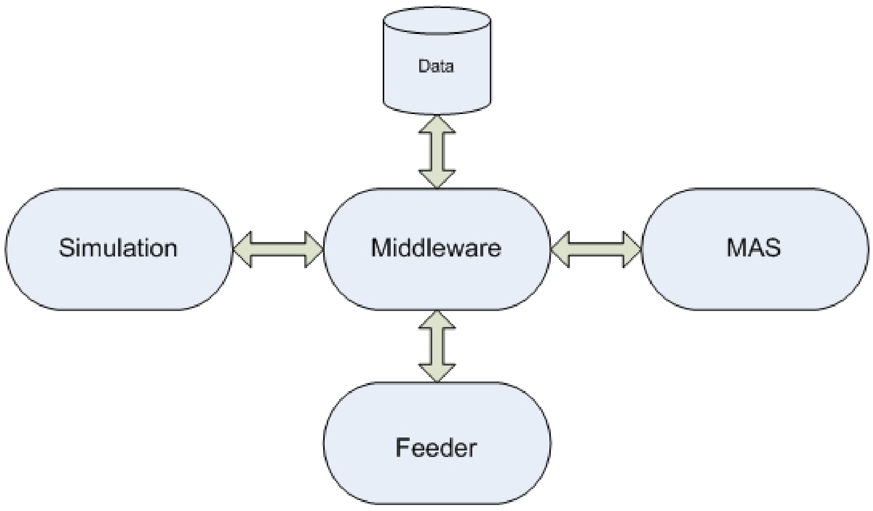


Figure 6.3: Complete System

6.3 Types of Middleware

There are various types of middleware that can be used to connect two different pieces of software. We can categorise typical middleware types as the following[2]:

- **Remote Procedure Call:** Client makes calls to procedures running on remote systems.
- **Publish/Subscribe:** This type of middleware monitors activity and pushes relevant information to subscribers.
- **Message Oriented:** Messages sent to the client are collected and stored until they are acted upon, while the client continues with other processing.
- **Object Request Broker:** This type of middleware makes it possible for applications to send objects and request services in an object-oriented system.
- **SQL-oriented Data Access:** Middleware between applications and database servers.

[2]http://en.wikipedia.org/wiki/Middleware

In the complete system shown in figure 6.3 we have a Middleware application connecting the simulation model with a multi-agent system, thus requiring two middleware types as shown in figure 6.4; one type connecting the simulation model to the middleware application, and another for connecting the multi-agent system with the middleware application. The middleware application can thus be seen as a translator translating data from the multi-agent system to a format, which can be understood by the simulation model and vice versa.

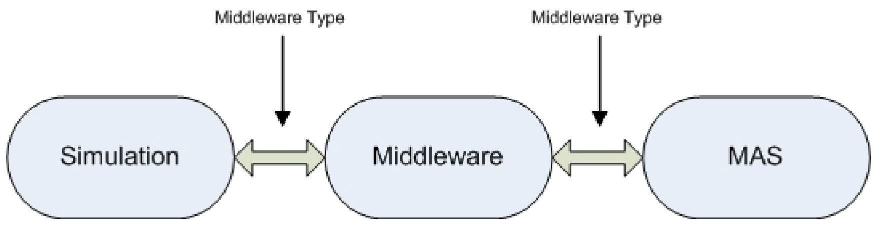


Figure 6.4: Middleware types

6.4 Middleware Type alternatives for AutoMod

The simulation environment in AutoMod can be accessed in two ways[3]:

1. Active X
2. Socket Communication

By ActiveX the simulation can be accessed directly from the middleware and it is possible to execute functions in Automod model from the middleware (see the ActiveX design in section 10.6). Objects can be sent between middleware and simulation, which also makes this approach very flexible.

By socket communication the AutoMod simulation model can be accessed with messages. A message can consist of the data formats, “String”, “Real” and “Integer”. The benefit of using sockets instead of ActiveX is that the middleware will not have to run at the same machine as the simulation model. The middleware and simulation model only needs to be connected by a network.

The ActiveX in Automod can be classified as a Object Request Broker type, because the middleware can call functions in the simulation which in turn

[3]A third alternative is OPC communication, which is used if there are OPC servers in the system that are going to be emulated. This is not the case in our thesis.

can return objects as a result to the middleware.

The socket communication in general can be classified as a Message oriented type, because messages are sent between two sockets using an underlying transportation protocol called TCP[4] or UDP[5].

We have chosen ActiveX as middleware type between the AutoMod simulation model and the middleware application, because it provides faster access, and the ability to access simulation model objects directly.

[4]Transmission Control Protocol
[5]User Datagram Protocol

Part III

Design

Chapter 7

Data structures

Data structures are used for representing data. There are data structures which are more suitable than others in specific situations, for example when developing a software dictionary a hash tabel is a good data structure to use [3].

This chapter presents methods for representing a graph [3], which can be used to represent the road system at OSS. The advantage of using a graph is that there exist a lot of well documented graph algorithms, that are useful for path finding. Furthermore we describe TAEMS, which is a language for structuring tasks.

7.1 Graph Representation

Informally, a graph is a finite set of points, referred to as vertices or nodes, some of which are connected by lines or arrows, called edges [9]. If the edges in the graph are undirected or bi-directional, the graph is called an undirected-graph or simply a graph. If the edges in the graph are directed or one-directional, then the graph is referred to as a directed graph or digraph. In general the term “graph” refers to both undirected and directed graphs.

In the OSS domain all roads are bi-directional, therefore we will use an undirected graph to represent the road system at OSS(described in section 9.3). The following formal definition of an undirected graph has been adapted from [9]:

Definition 7.1 Undirected graph
An undirected graph is a pair $G = (V, E)$, where V is a set whose elements are called vertices, and E is a set of unordered pairs of distinct

elements of V. Vertices are often also called nodes. Elements of E are called edges, or undirected edges for emphasis. Each edge may be considered as a subset of V containing two elements; consequently, v, w denotes and undirected edge. In diagrams this edge is the line v—w. In the text we simply write vw. Of course, vw=wv for undirected graphs.

Furthermore roads in at the OSS domain have lengths referring to their distances, which should be represented in the graph. This can be represented using a so called weighted graph. The following formal definition of a weighted graph has been adapted from [9]:

Definition 7.2 Weighted graph
A weighted graph is a triple (V, E, W) where (V, E) is a graph (directed or undirected) and W is a function from E into **R**, *the reals. (Other types for weights, such as rationals or integers, may be appropriate for some problems.) For an edge e, W(e) is called the weight of e.*

Figure 7.1 shows a weighted undirected graph, which is a sub-graph of the graph shown in figure 9.4. The nodes 1 to 8 in figure 7.1 corresponds to the nodes S811_1, V111_1, V111_2, U211_1, U316_1, V122, U316_2 and V211_1 in figure 9.4 respectively.

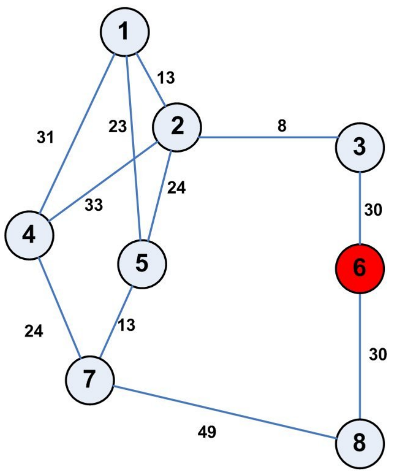


Figure 7.1: A weighted graph. Subpart of the graph in figure 9.4.

There are two standard ways of representing a graph: as an adjacency matrix or as a collection of adjacency lists, which are both applicable on directed and undirected graphs. Typically one will use adjacency lists when

	1	2	3	4	5	6	7	8
1	0	13	∞	31	23	∞	∞	∞
2	13	0	8	33	24	∞	∞	∞
3	∞	8	0	∞	∞	30	∞	∞
4	31	33	∞	0	∞	∞	24	∞
5	23	24	∞	∞	0	∞	13	∞
6	∞	∞	30	∞	∞	0	∞	30
7	∞	∞	∞	24	13	∞	0	49
8	∞	∞	∞	∞	∞	30	49	0

Table 7.1: The adjacency matrix of the graph in figure 7.1

dealing with sparse graphs, where $|E|$ is much less than $|V|^2$, because it requires less memory than an adjacency matrix. An adjacency matrix is preferred when the graph is dense, i.e. $|E|$ is close to $|V|^2$. Table 7.1 shows the adjacency matrix for the graph in figure 7.1 and figure 7.2 shows the adjacency lists.

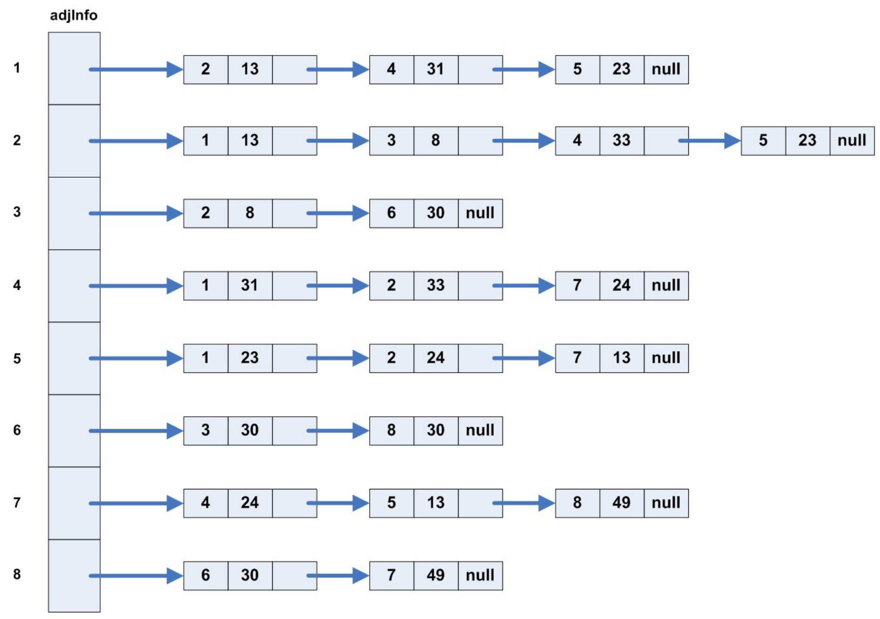


Figure 7.2: The adjacency lists of the graph in figure 7.1

7.2 TAEMS

Task Analysis, Environment Modeling and Simulation (TAEMS) is a modeling language for describing the task structures of agents and is used as a framework for environment centered analysis and design of coordination mechanisms [31]. The representation of agent activity should make it possible for multiple individual agents to contribute to independent and different high-level goals.

An agent must have a representation of what its capabilities are, and the agent must be able to reason about its potential actions in the context of its working environment. TAEMS provides a task structure describing the tasks an agent can perform. TAEMS represent a wide range of ways a particular task can be performed, by adding features to conventional task structures, such as quantitive action characterizations, and explicit models of local and remote interactions and mechanisms.

Even though the details associated with TAEMS is very comprehensive, its structure and function are actually based on a few simple concepts. The TAEMS task structure is a commented task decomposision graph, but it is referred to as a tree for simplicity.

- Highest level nodes : are the task groups, which represents goals that an agent may try to achieve.
- The level below the task groups : is a sequence of tasks and methods describing how the corresponding task group may be performed.
- Tasks : represents sub-goals, that can be decomposed further.
- Methods : are terminal (leaves in the tree), which represent the actions an agent can perform.

Tasks contains annoations that describe how sub-tasks may be combined to satisfy the given task. Interrelationship is another type of annotation, which describes how achievement of goals or execution of methods affects other nodes in the structure. There exist several interrelationships that describes various situations. We can have several task tree structures with interrelationships between them, called non-local interrelationships, which indicates agents having knowledge about other agents capabilities. The non-local interrelationships implicit describe situations where negotiation or coordination may be desired.

In figure 7.3 an example of a TAEMS task structure is shown, containing a task group, which consists of three tasks and their corresponding methods and interrelations.

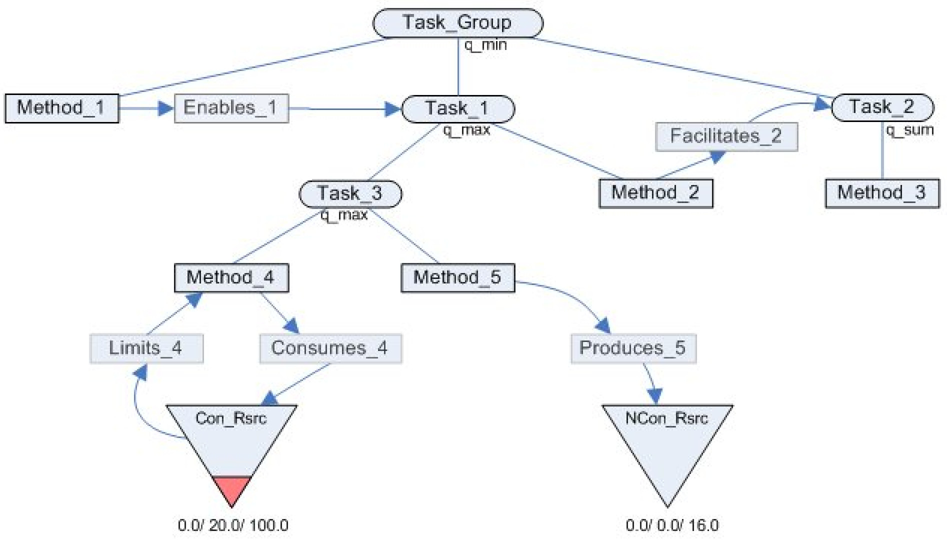


Figure 7.3: An example of a TAEMS task structure

Enables relationship

An enables interrelationship is a hard variety of a interrelationship, in that sence that if a task enables a method, that method is not accesible until the task have been completed. Thus, if a task, executing a method M_1, enables method M_2, then M_2 can not run until M_1 is completed.

As an example, let us assume we have two tasks; task A, which concerns bying some food, and task B, which cencers eating some food. If we do not have any food, then task A, enables task B, meaning that we can not eat, before we have bought some food, thus enables is considered a hard relationship.

Facilitates relationship

A task A can facilitate another task B, meaning that task A for instance descreases the duration of processing of task B, or that it increases the quality of task B. Facilitates is a soft relationship in the sence that task B can execute even if task A has not yet been executed.

As an example, if we say we have a task A, which is heating some food in a microwave, and a task B, which is eating the food, then we could certanly eat the food (execute task B), without heating it first (performing task A), but if we warm up the food before we eat it, then the outcome of eating, would result in greater satisfaction, thus facilitates is considerd a soft relationship.

Hinders relationship

Hinders is the opposite of facilitates, meaning that if task A hinders task B, then task A will increase duration of task B and/or decrease the quality of task B.

As an example, consider a married couple driving to visit some friends, if we say that task A is the man driving in order to reach their destination, and task B is the wife giving bad directions, then performing task B, will result in increasing the duration of task A. Hinders is, as facilitates, a soft relationship.

Precedes relationship

Precedes is a combination of enables and hinders, so if task A precedes task B, then task A must finish before task B and task A must be done well, or later tasks(including task B) will suffer from it.

Favor relationship

The favor relationship is actually not a new kind of relationship, but can be any relationship, which has a positive effect.

Chapter 8

Algorithms

A good path-finding algorithm is essential to artificial intelligence, thus this chapter present several algorithms which can be used for path finding. Furthermore coordination of multiple agents in a system requires some sort of coordination algorithm, which is why we have described some strategies and algorithms.

8.1 Path-finding basics

Path finding can be reduced to getting from point A to point B [12]. A path from one point A to another point B can potentially have different solutions, but ideally we want a solution that solves the following goals:

1. How to get from A to B
2. How to get around obstacles
3. How to find the shortest possible path
4. How to find the path quickly

There are path-finding algorithms which solve all of the above problems and algorithms that solves none of them, for example if one wishes to travel from Denmark to Canada by train, then there exist no solution, unless you can get a ship too sail to Canada with the train.

8.2 Breadth-First Search

Breadth-First Search (BFS) involves visiting nodes one at a time in a graph [12, 3]. The nodes are visited in order of their distance from the source node, where the distance is measured as number of traversed edges. Nodes one edge away from the source node is thus investigated first, and so on until all nodes are visited on the way to the goal. This method ensures that you find a path from source to destination with minimum number of edges. Another way of explaining this, is to visit all your neighbors first, then visit all your neighborś neighbor nodes, and continue this process until the goal is found. Figure 8.1 shows an example of the breadth-first search algorithm, wherein the nodes are numbered based on the search succession.

Too avoid obstacles with BFS make sure that no obstacles are connected to any nodes, but nodes are only connected were it should be possible too travel.

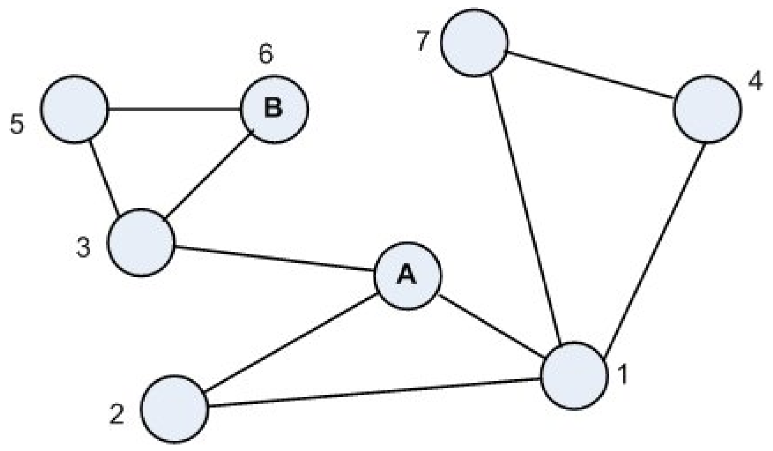


Figure 8.1: An example of breadth-first search, and the nodes are numbered in the order of the search

To keep track of all the nodes, We put all visited nodes in a "closed" list, and the nodes we intend to visit in an "open" list. If we encounter a node that is already in the closed list we simply ignore it, thereby avoiding visiting the same node twice. The open list is a First In First Out (FIFO) list, meaning that the first nodes in the list are also the first nodes we look at.

8.3 A* Search

The A* –pronounced "A-star"– search algorithm is an extension of the breadth-first algorithm [12], which is already integrated widely as part of the artificial intelligence in several computer games. Following extra factors are included in the A* search algorithm:

- Edges are given different "cost", which indicates the cost of traveling from one node to another.

- The cost from any node to the goal node can be estimated, which helps refine the search, such that we are less likely to search in the wrong direction.

The cost between the nodes do not have to be the distance, it could also be the time it takes to travel from one destination to another. We can imagine roads on which it takes longer time to travel even though the distance is shorter, for example in places with lots of traffic or narrow places, where we have to drive with caution, thus decreasing the speed of the vehicle and increasing the traveling time. The A* algorithm works the same way as BFS, except for these changes:

- The nodes in the open list are sorted by the total cost from the start to the goal node, thus it is a priority queue. The total cost is the sum of the cost from the source node to the goal node.

- A node in the closed list can be move back to the open list if a shorter path (less cost) to that node is found.

The open list is now sorted by the estimated cost instead of the FIFO method in BFS, which means that it searches the nodes that are more likely to be in the direction of the goal. Figure 8.2 shows an example of an A* search in a graph, where node A is the start location and node H is the destination. In each node the estimated cost is specified (the estimated cost in this case is the direct distance to node H), e.g. the estimated cost from node B to the destination node H is 300. The red spots indicate the sequence of the nodes that are analyzed when the algorithm searches for the shortest path from node A to node H (See section 14.4.4 for implementation specific details about the A-star algorithm).

8.4 Dijkstra's algorithm

Dijkstraś algorithm is a greedy algorithm that solves the single-source shortest path problem for a weighted directed graph G = (V, E) with non-negative edges $w(u, v) \geq 0$ for each edge $(u, v) \epsilon E$ [3].

The algorithm maintains two sets of nodes S, which starts out empty and Q, which starts with all nodes in V. The set S contains all nodes, whose final shortest-path weights from the source s have already been determined, and the set Q contains all the other nodes. When the algorithm is done all nodes will be in the set S, and the set Q will be empty.

Dijkstraś algorithm is greedy, and thus it repeatedly selects the node $u \epsilon V - S$ with the minimum shortest-path estimate, then adds u and relaxes all the

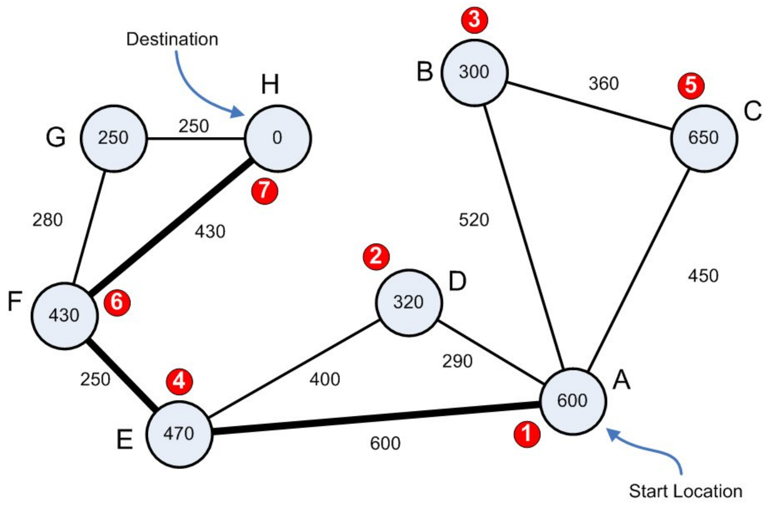


Figure 8.2: An example of A* search. All edges have a cost, and the most thick path indicate the shortest path from A to B.

edges leaving from u.

The algorithm works by holding the cost d[v] of the shortest-path found so far for between the source s and the node v. Initially, the value is 0 for the source node s (d[s]=0), and infinity for all other vertices, which represents that we do not know any path leading to those nodes, thus d[v]=? for each node v ϵ V, except s. When the algorithm is finished, d[v] equals the cost of the shortest path from the source s, which will be $infty$ if there exists no path to that node.

Dijkstraś algorithm makes a minimum spanning tree (MST), which a lot-similar to the MST created with the PRIM algorithm, the difference is that the spanning tree spans from a single source, which we specify, and calculates the shortest distance to all nodes from that source. The figure 8.3 (figure 24.6 from [3]) shows the execution of Dijkstra's algorithm on a graph with 5 nodes and 10 edges.

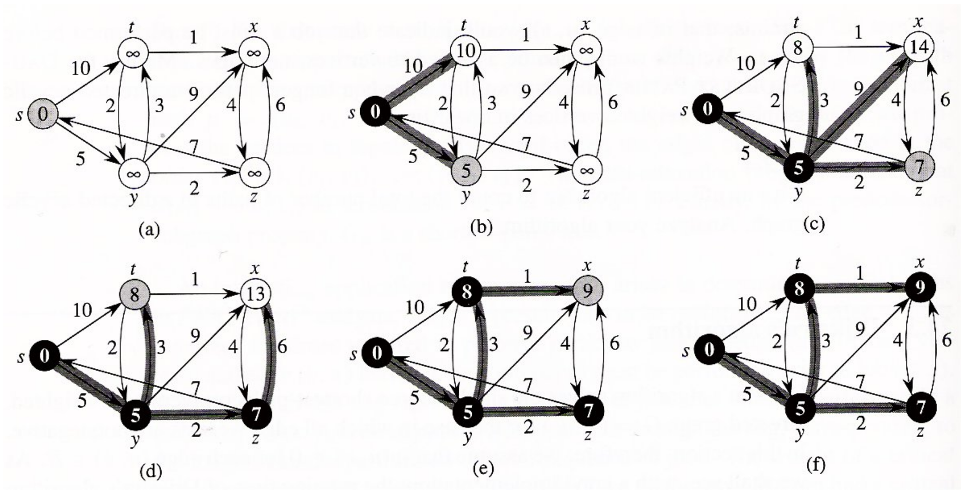


Figure 8.3: Example of Dijkstraś algorithm. The shortest-path estimate are shown in the nodes, and shaded edges indicate predecessor values. Black nodes are in set S, and white nodes are in set Q = V-S. The last figure f shows the value d for every node.

8.5 Generalized Partial Global Planning

This section concerns Generalized Partial Global Planning (GPGP), which is a family of generic coordination mechanisms for coorporative, soft real-time computational task environments [16]. GPGP form a basis set of coordination mechanisms for teams, which consist of cooperative agents.

Partial Global Planning (PGP) do not handle deadlines, thus PGP is not suitable when dealing with systems with real-time constraints. In "real-time" problem solving agents may have goals with strict deadlines, which ads constrains to the plan activity. Planning is often reactive in real-time situations, rather than reflective, where a sequence of actions is planned out in some detail prior to execution. The reason for reactive planning is that the agents should be able to respond quickly to changes in the environment, and because the outcome of the actions might be too uncertain when planning too far in the future.

There are three basic areas of agent coordination behaviour specified with GPGP:

- Consider a current problem situation, when and how do we construct and communicate non-local views.
- When we have partial results of problem solving, when and how do

we exchange those results.

- how and when to make and break commitments to other agents about what results will be available and when

In the following sections we describe how GPGP covers the areas mentioned above, with a local scheduler and a family of generic coordinating mechanisms.

8.5.1 The Local Scheduler

Every agent contains a local scheduler, that produces a schedule of what methods to execute and when. The local scheduler takes the agents subjective believed task structure as input, which contains information regarding potential duration,potential quality and the relationships between methods, such that the scheduler can choose and order the executeable methods. This is used maximizes to maximize a pre-defined utility measure for every task group T, where the utility function is the sum of the task group qualities.

The scheduler accepts a set of commitments C from the coordination component. Commitments is used as contraints on the schedules produced by the local scheduler. If we for instance have two agents; agent A, which executes method 1, and agent B, that executes method 2, then if the methods are redundant, agent A could make the commitment to agent B, that it will execute the method and share the result with agent B. Commitments between agents are also called non-local commitments (NLC).

More than one schedule may be produced by the local scheduler as result of the scheduler trying to satisfy the set of commitments when not all commitments can be met. A function Violated(S) called with the schedule S, returns a set of commitments that are believed not to be met in that schedule.

An ideal local scheduler would produce a schedule with a maximum utility measure and a schedule with maximum utility that satisfies all commitments, but in practice a set of schedules are produced where the utility measure is not neccesarily optimal.

8.5.2 GPGP Coordinating Mechanisms

Generalized partial global planning contains five coordination mechanisms; updating non-local view-points, communicating results, handling simple redundancy, handling hard coordinating relationships and handling soft coordinating relationships.

Mechanism 1: Updating Non-local View-points

Agents - in the context of GPGP, only have a partial, subjective view of the present episode. Partial views can be increased by sharing non-local information, and agents can even communicate all their private structural information in an attempt to create a global subjective view.

This mechanism can be used for updating non-local views among agents, by communicating all private information, some information, or no private information. When choosing to communicate some information, a partal view that is, then the agent only communicates information, that are related to another agent by a coordination relationship. A information gathering action called "detect-coordination-relationships" is used to detect coordination relationships between private and shared parts of task structures, when a new task structure arrives.

Mechanism 2: Communicating Results

This coordinating mechanism has three possible policies:

- Minimal Policy : Communicate only the results neccesary to satisfy the commitments (local view).
- Task Group Policy : Communicate the results like with minimal policy, and the final result of a task group (partial view).
- All policy : Communicate all results (global view).

Mechanism 3: Handling Simple Redundancy

An example of a GPGP coordination mechanism is one that handles simple method redundancy. If more than one agent has an otherwise equivalent method for accomplishing a task, then an agent that schedules such a method will commit to executing it, and will notify the other agents of its commitment. If more than one agent should happen to commit to a redundant method, the mechanism takes care of retracting all but one of the redundant commitments

Mechanism 4: Handling Hard Coordinating Relationships

Hard relationships include relationships like enables, which was described in section 7.2. The direction of the hard coordination relationship is further distinguished, meaning that currently only the predessor is aware of the relationship.

Mechanism 5: Handling Soft Coordinating Relationships

Soft relationship include relationships like facilitates and hinders, which was described in section 7.2 and 7.2 respectively. At the moment the facilitates relationship is directed, and only the predessor of the relationship is aware of the relationship. The hinders relationship, should opposite the facilitates relationship be placed at the successor.

Chapter 9

Solution Strategies

In this chapter we propose solution strategies for solving the transportation problem at OSS based upon multi-agent technology.

Our problem can be divided into two main problems:

- Transporting: Transporting a ship block from one location to another using a KAMAG vehicle.
- Coordinating: Coordinating transportations, i.e. make a logistic plan regarding what KAMAG should be used to transport a certain block, and at what time.

9.1 Organization of agents

A central discussion in a multi-agent system is where to place the agents. There are many possibilities. In this section we describe where we have placed our agents and how that will effect the organization, interaction and environment in which the agents reside. We have considered alternative agent placements, and will cover those briefly.

In the research phase, wherein the problem domain was investigated, we implicitly categorized the actors in context of the daily transportation at OSS, which is exactly where we place our agents.

Actors -in context of the problem domain, include the following:

- C-planners
- D-planners
- Kamag vehicles

As mentioned earlier C-planners requests transports from the D-planner, which coordinate transports and delegates transportation tasks to kamag vehicle drivers. The geographical placement of the various C-planners can be seen in figure 9.1, which includes C-planners from B4, B6, B9, the paint halls, the south hall and the east hall. We will model the C-planners as Planner Agents (PAs), and from now on refer to theese agents as PAs. We model the D-planner as a Coordinator Agent (CA), and this agent will from now on be referred to as the CA. The Kamag Vehicles, will be modelled with Kamag Vehicle Agents (KVAs), and will from now on be referred to as KVAs. The relation between the CA, PAs and the KVAs can be seen on figure 9.2.

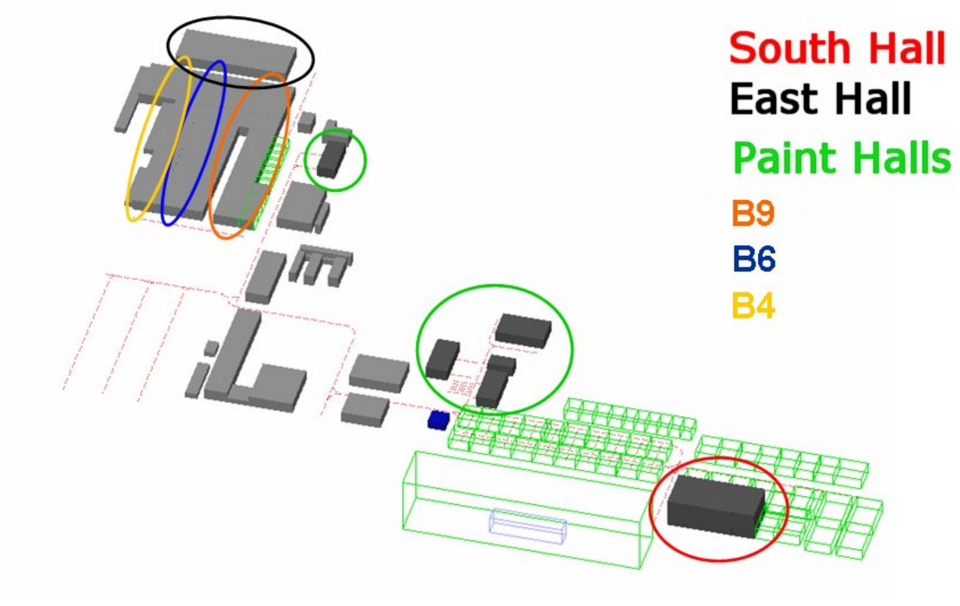


Figure 9.1: Geographical placement of various C-planners

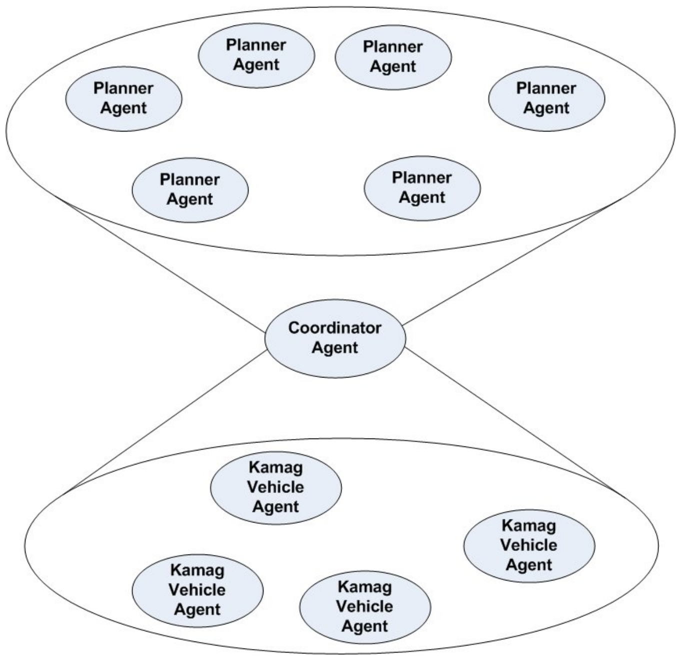


Figure 9.2: Agent Organization showing the relation between PAs, the CA, and the KVAs. Planner agents can communicate with each other, and with the CA. The CA can communicate with any PA and any KVA, and each KVA can communicate with each other.

The agent organization clearly indicates that the human planning hierarchy have been preserved, which will impact the agents subjective perception of their environment, their interaction, and organization.

9.1.1 Planner Agents

Planner agents are human controlled, thus we have designed a graphical interface, that allows planner agents to plan and replan transportations, and give them a dynamically updated view of the consequences of their transportations, such as estimated arrival at pickup location and destination location of a given transportation. Planner agents request daily transportations of ship blocks, that should be transported from one location to another location at a specific deadline. Transportations are given a priority, that indicates how important the task is.

Priority 1 : the highest priority, which can only be given to transportations to the gantry crane, indicating a very strict deadline.

Priority 2 : indicates that this transportation is urgent.

Priority 3 : indicates that this transportation is not urgent, meaning a non-strict deadline.

9.1.2 Kamag Vehicle Agents

A KVA should have different mechanisms it can use to solve different subtask, such as

- Route-planning: calculating a path from one location to another.
- Driving: follow a specific route, from one location to another
- Perceive: View of all locations which spans X locations from current location.
- Estimating time: time estimating a given task, such as estimating how long it will take to arrive at some destination.
- Block placing: picking up a ship block at its current location or putting down a ship block at its current location.
- Communication: Receive tasks. Sending and receiving states, routes, results and estimates.

Each KVA have a perception, which is used to view X locations ahead from its current location. By using their perception, KVAs are able to detect each other and obstacles in the environment. When two KVAs perceive each other, they exchange routes in order to calculate if their paths will intersect, and it is up to the agent to be proactive and or social in order to solve possible conflict. A conflict, meaning a route intersection between two KVAs will require negotiation and relocation of one of the KVAs. Figure 9.3 illustrates the perception of a KVA in its environment, see section 9.3 for a description of the agent environment.

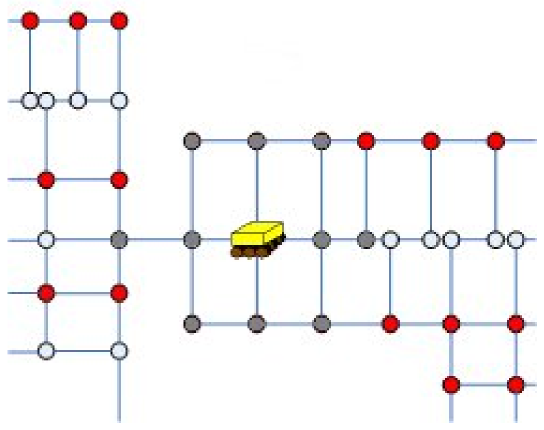


Figure 9.3: Example of KVA perception, where all locations within a distance of two locations from the KVA are included. The grey nodes(loactions) indicate the KVA perception

9.1.3 The Coordinator Agent

The CA should have mechanisms for receiving plans from PAs, to coordinate the plans into schedules, delegating tasks to KVAs and to update the schedules dynamically when changes occur. The CA should have a mechanism for dynamic planning and replanning with Kamag vehicle agents. If a certain transportion request from a PA can not be satisfied by any KVA, the CA should report this information back to the PA, such that the PA have the option of changing a request.

The strategy used when coordinating the transports is to sort the transport request in assending order with repect to the deadline set on the request, when received from planner agents. The coordinator then forwards the transportation request to the KVAs, which then reports a Bid on when they solve the task, or a failure notice if it is not posible to solve the task, and the KVA with the best bid is delegated the task.

9.1.4 Alternative agent placements

A possibility of placing agents different from our solution, could be placing agents at every control point(location), which should have the responsibility of navigating travelling vehicles around the road system, much like how routers navigate packages around within the internet.

One could also have placed agents at every storage location, making storage agents negotiate on where to place a given shipblock.

If we look at an abstraction level higher, taking the B-planner into account at the shipyard, then it would be possible to model the entire flow of every shipblock through the system, and it would resemble a traditional production flow control system. The various B-planners should then be organized as agents, and then the C-planners would be fully automated negotiating with both the B-planners and D-planners at the shipyard (see sections 2.2.2, 2.2.3 and 2.2.4). When looking at this abstraction level, it would make sence to place agents in each shipblock, since the shipblock agents then could have knowledge regarding their destined flow in the system, and it would be possible for the shipblock agents to request different services from other agents, such as the transportation service like we have modelled, and futher to use supply, equipment and paint agents.

Summary of agent placement

There exist endless posibilities on where to place agents in a system, and it is thus in the hands of the developers that designs the systems. We have choosen to place our agents, where the human actors naturally reside in the problem domain, in order to preserve the human organization.

9.2 Agent Interaction

As described in section 9.1, the PAs sends information to the CA holding transportations requests, the CA coordinates theese request, and delegates tasks to KVAs.

How the agents communicate is highly dependent on what multi-agent framework is being used, thus we refer to chapter 12 for communication in the Cougaar framework and chapter 11, which descripes agent interaction with Decaf.

9.3 Agent Environment

We represent the Agent environment at OSS with a graph data structure as seen in figure 9.4[1]. The nodes form together with the edges the road system, where KVAs can travel. There are placed nodes at every intersection between roads. There can only travel one KVA on a road (edge) at a time. We have colored the graph to give a better overview of the placement of storage locations, painting halls, production and equipment halls, and

[1]If you are reading the pdf version of this thesis, you can zoom in at any node to see its name

parking lots.

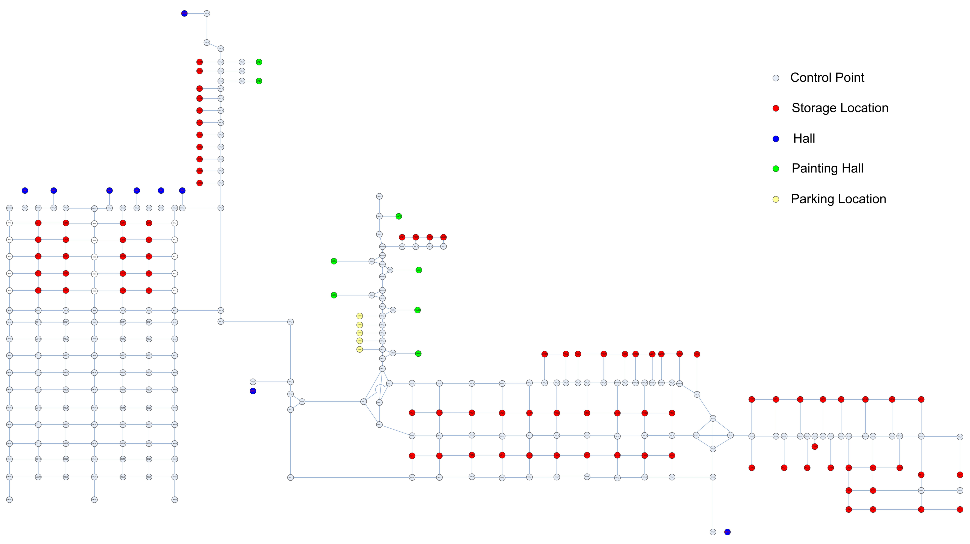


Figure 9.4: Graph representation of the Agent environment. All nodes are control points where Kamag vehicle Agents (KVA) can travel between. Red nodes indicate storage locations. Green nodes indicate painting halls. Blue nodes indicates production and equipment halls. Yellow nodes indicate parking lots for KVAs.

The costs on the edges are omitted on figure 9.4 in order to simplify the overview of the graph. The real distances between each node have been calculated with the use of Automod, which have procedures to calculate distance from on location to another. The distances have been put in the graph data structure, and we have named the nodes so that they comply with the naming conversion used in Automod for simplifying communication between the multi-agent system and the simulation model.

If we take a look at the class diagram in figure 9.5, we get an idea of the association between nodes, edges, KVAs, ship blocks, obstacles and queues. The class diagram have been made with OO notation and is used to indicate an abstract graph entity association, which will be mapped into a multi-agent design approach later on .

- Nodes : Nodes are all locations, where a KVA can be at, and travel between, we also refer to theese locations as control points (CP).
- Edges : Edges are roads that connects nodes, there can be one or more roads entering or leaving a node.

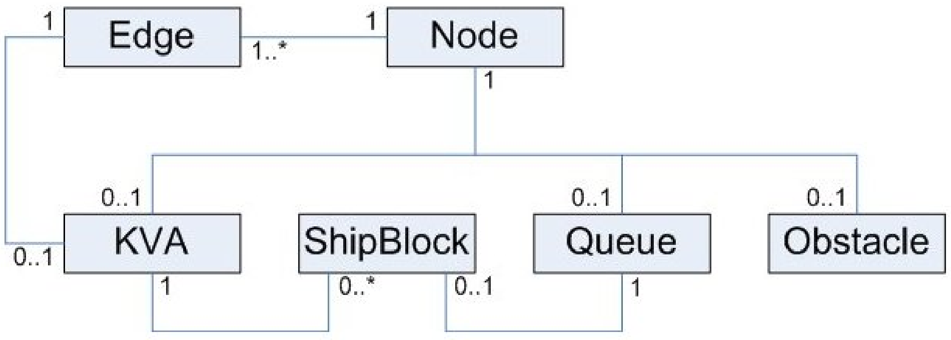


Figure 9.5: Class diagram showing associasions between entities in a graph

- KVA : There can be at most one KVA on a road or at a control point at a time. A KVA can carry multiple ship blocks.
- ShipBlock : A ship block can either be on a KVA or in a queue.
- Queue : A queue is a container for ship blocks. A queue can have different properties, such as storage, paint, production, equipment and supply.

9.4 Finding The Shortest Path

Finding the shortest path from one location to another is important in order to solve various subtasks, thus we have designed an algorithm, which based on the A* search algorithm finds the shortest path from one control point to another control point (see control point placement at figure 9.4). The function findPath(o1, o2) takes two control points as input and returns the shortest path in form of a list with the control points the vehicle must travel to reach its destination.

9.5 Find The Nearest Location

An agent will experience several situations where it will come in handy to know the location of the nearest free control point, that is not in a specific route, for example when KVAs have to move out of each others way. A dilemma is seen on figure 9.6 where KVAs will have to negotiate in order to continue on their route.

We have designed a function called getNearestLocation(o1, o2), which takes a route and a control point(CP) as an input and returns a control point (location). The function creates a minimum spanning tree (MST) from a source

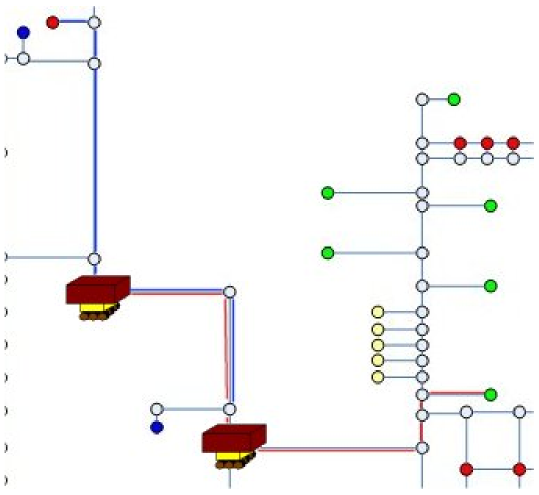


Figure 9.6: Example of path interference between two KVAs route. The route of the first KVA is red, and the route for the second KVA is marked with blue color

location, where every control point is placed in a list (visitList), the list is sorted in ascending order. We run through the visitList finding the first control point that is free and not in the route list.

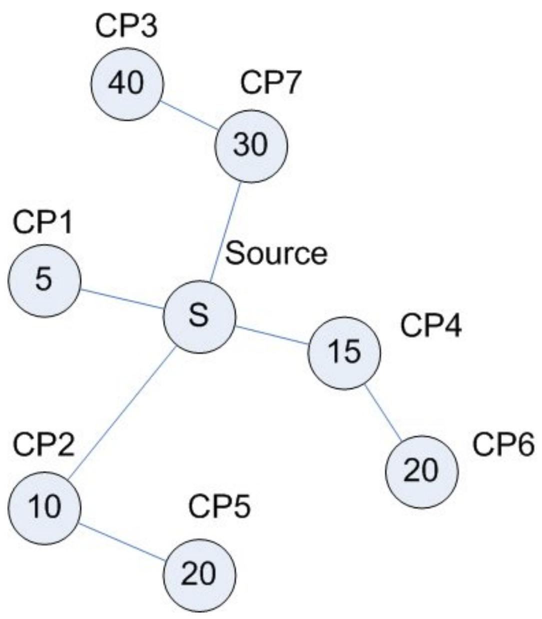


Figure 9.7: Example of a minimum spanning tree. The cost is written in the nodes, and the nodes are random numbered, from CP1 to CP7, the source node S indicates where the tree spans from

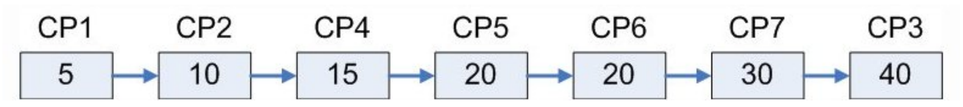


Figure 9.8: Control point from figure 9.7 sorted in ascending order

1. Calculate MST
2. Sort control points in ascending order
3. test: is CP free and not in the route list
4. return found CP or null if no free CP could be found

9.6 Conflict Resolution

When to KVAs needs to use the same route simultanously they have to negotiate in order to resolve, which KVA that gets to pass through first, and which KVA that must wait or find another temporary location, while the othe other KVA pass through. Figure 9.6 illustrates a scenario where two KVAs have intersecting routes, and chooses to be social in order to resolve the conflict. Figure 9.9 shows the scenario after the negotiation where the KVA with the blue route have moved out of the way for the other KVA (with the red route). Figure 9.10 shows that KVA 1 perceives that KVA 2 has passed through its route, and thereby proceeds on its route. The conflict have been resolved.

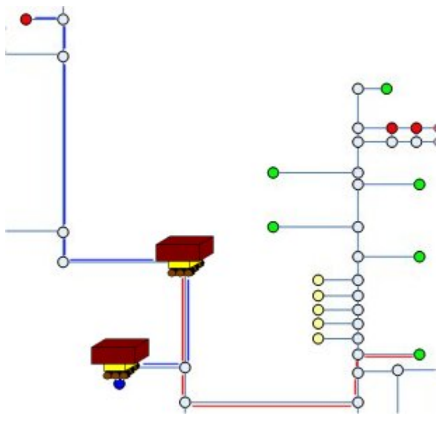


Figure 9.9: Conflict solving with socializing. The KVA with the blue route, redirects to the first location not in the route of the other KVA(red route)

.

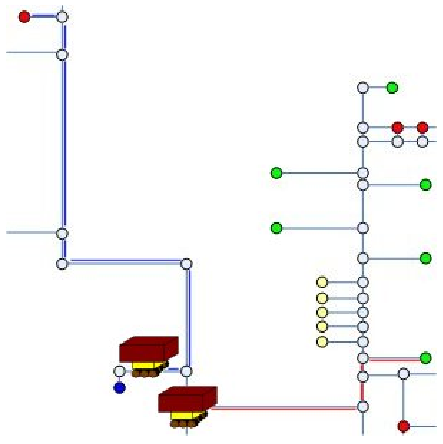


Figure 9.10: Conflict solving with socializing. The KVA with the red route has passed the KVA with the blue, which perceives this in its environment and continues on its route.

The strategy we use to avoid KVA collision is that when a KVA perceives another KVA, it socializes with the other KVA, exchanges routeplans, together they determine if their routes will intersect. If their routes intersect they compare priority, the KVA with the lowest priority will move out of the way. If both KVA transportations have the same priority, the KVA which has the shortest path to a free location, will move out of the way.

9.7 Coordination of transports

The transportations at OSS are categorized into priority and deadlines, e.g. ship blocks that are to be delivered at the gantry crane has the highest priority. Therefore the overall strategy used to coordinate the transportations of ship blocks, is to sort and delegate the transportations in respect to their priorities and deadlines. How the coordination will be performed is highly dependent on the planning capabilities of the MAS framework being used. Planning capabilities used for coordination purposes are for example utility functions and GPGP, which were described in section 8.5.

9.8 Summary

We have proposed solution strategies to the logistic planning problem at OSS in terms of multi-agent technology.

- **Agents:** We have proposed a solution, where we use
  - reactive planning agents, that are controlled by humans(C-planners). The planning agents sends transportation requests from a GUI, and recieves dynamical notification regarding the status of each transportation request, such that it is possible to replan, when deviations from the schedule plans occur or the plans are not possible to satisfy.
  - a coordination agent which is reactive in the sence that when new requests are received it delegates the task to best biding KAMAG vehicle agent, proactive in the sence that it makes a logistic plan, that describes what KAMAG vehicle agent that will perform a given transportation request and when. Thus the logistic plan is a coordination of requests and resources(vehicles).
  - a KAMAG vehicle agent, that is reactive in the sence that it dynamically changes its route if it perceives an obstacle on its route. It is proactive in the sence that it plans the sequence of a transport, regarding calculating an entire route from one location to another, and when to load/unload ship blocks. Furthermore it calculates when to start driving in order to reach its arrival destination at a deadline. The KAMAG vehicle agent can accept or deny a transportation request, depending on its status, schedule and its location[2] at the start time of a transportation request.
- **Environment:** We have proposed a solution based on a graph(which is tightly coupled to the designed AutoMod path movement system), where nodes represent locations a KAMAG vehicle can drive to. Nodes are placed in each storage location, at every hall, and furthermore on every road intersection. Edges in the graph represent roads with distances, which are the real world distances, obtained from the designed AutoMod path movement system. Each node can contain a KAMAG vehicle and/or a ship block, which is useful when KAMAG vehicle agents perceives their environment.
- **Interaction:** We have described that interaction between agents in a multi-agent system is highly dependent of the MAS framework being used, hence interaction must be adressed according to the used MAS framework.

[2]The location it will be at the time that it should start execution the task

- **Organization:** We have proposed a possible solution for organizing the agents into three types; planners(C-planners), coordinators(D-planners) and KAMAG vehicle agents(KAMAG vehicle drivers), where the coordinator receives transportation request from planners, and delegates those to KAMAG vehicle agents. By using this organization, we have preserved the hierarchical organizational structure at OSS.

To avoid collisions with other KVAs we proposed a solution strategy, based on agent perception and negotiation, more specifically a perception based on using a MST[3] combined with the graph(the environment), making a KVA able to detect other KVAs within its perception. When other KVAs are detected they should socialized and use the following strategy to avoid collisions:

1. Socialize: has the following sequence:
   (a) exchange route plans
   (b) check if routes intersect. If yes then negotiate, else stop socializing
2. Negotiate: has the following sequence:
   (a) calculate route to the nearest free temporary location that is not in the other KVAs route
   (b) compare transport priority and act accordingly:
      - different priority: The KVA with the lowest priority moves to its temporary location.
      - same priority: The KVA with the shortest path to the temporary location moves to that location.

[3]Minimum Spanning Tree

Chapter 10

Simulation Model

This chapter will describe the functional requirements to the simulation model and how we have designed our simulation model with AutoMod, as seen in figure 10.1, in order to fulfill those requirements. We will only cover a sub-part of AutoMod, which concerns the elements necessary to make a model of the OSS problem domain. Every step in designing the model will be explained, such that it should be possible for other students to make a similar model in AutoMod, which fulfills their requirements.

First we will list the functional requirements to the model and then explain which elements are typically involved in an AutoMod model, and finally give details regarding what elements to use in AutoMod in order to model the OSS problem domain, which is a sub part of AutoMod's functionality. Next we will explain how to decide, what elements to place where, for instance, how we decide to place a certain storage location in our model and how buildings are modeled.

10.1 Functional requirements

We have the following functional requirements to the simulation model:

1. Create ship block : It should be possible to create new ship blocks in order to simulate the production of new steel section, and the arrival of ship blocks from other countries.

2. Remove ship block : It should be possible to remove ship blocks from the system to simulate when a block leaves the system (from the dock) or when a ship block have been incorrectly created.

3. Storage of ship blocks : It should be possible to keep ship blocks at storage locations.

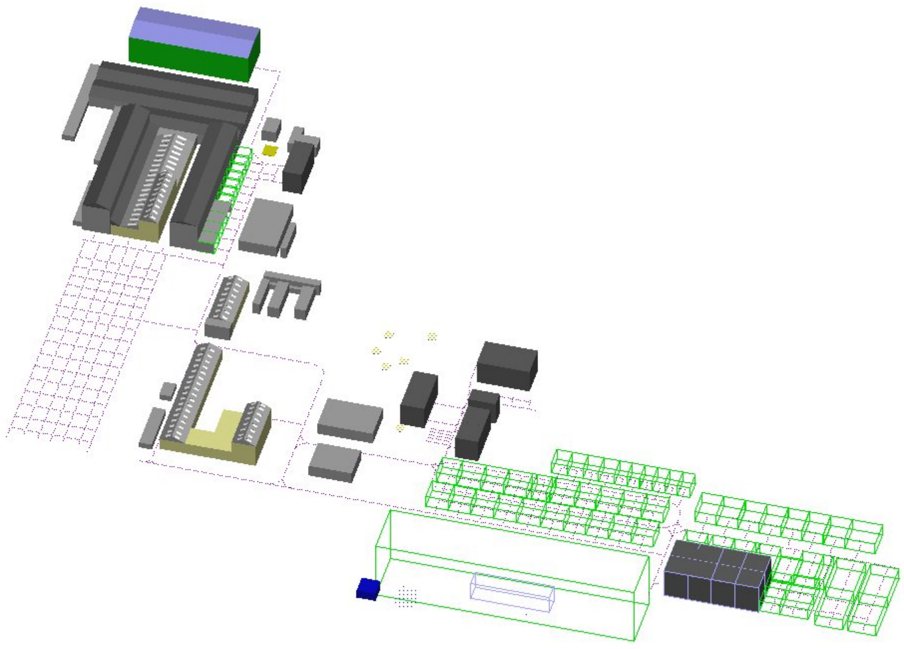


Figure 10.1: The final Simulation Model of the OSS domain. Storage locations, buildings, KAMAG vehicle and shipblocks are modelled.

4. Road system : It should be possible to model to road system.
5. Transport ship block : It should be possible to transport a ship block from one location to another location. This requirement will be further divided into sub requirements:
   (a) drive to destination: drive vehicle to a location - with or without a ship block.
   (b) pick up: It should be possible for a vehicle with the same location as a given ship block, to pick up that ship block
   (c) drop down: It should be possible for a vehicle at a given location, carrying a ship block, to place that ship block at that location.
6. External control : It should be possible to control the simulation model with an external system.

10.2 Elements in a manufacturing system

There exist two types of elements in a manufacturing system: permanent elements such as people, machines, buildings and material handling systems, and temporary elements that are manufactured, processed and later removed from the system, which are products that move through a system. Both types of elements can be modeled with AutoMod [45].

In an AutoMod model, permanent elements such as machines, people and buildings (such as painting halls) performing work on a product, are called resources; a robot is examples of a machine which can be modeled as a resource.

Temporary elements, are products which travel through the system, thus requiring a movement system in an AutoMod model; examples of available movement systems in AutoMod are: path mover systems, conveyors, Automated Storage and Retrieval Systems (AS/RS), power & free systems, bridge crane systems, tanks and pipes systems, and kinematics. The mentioned movement systems have the flexibility to simulate many real world systems such as baggage transportation systems in airports using conveyors.

10.3 Modeling the physical elements

This sub section will concern how to model the physical elements at OSS in AutoMod. We will explain how to model the following elements:

- Ship blocks
- Storage locations
- Buildings
- The road[1] system
- KAMAG vehicles

10.3.1 Ship Blocks Design

The ship blocks at the shipyard move through the system, and are thus temporary elements in our simulation model. Ship blocks are either delivered from subsuppliers or manufactured at the ship yard, at various locations and buildings, and are processed multiple places, thus these products are modeled with “loads” in AutoMod.

Physical entities that move through a system, are represented by loads in AutoMod. These loads are temporary elements, which are the active entity

[1]the word path and road refers to the same entity

Attribute/ Load	L_block	L_dummy
blockNo	√	
blockWeight	√	
blockHeight	√	
blockLength	√	
blockBreadth	√	
grandBlockNo	√	
putDown	√	
pickUp	√	
blockFamily	√	
transportTo	√	
transportFrom	√	
transportNext	√	
kamagPtr	√	
eventTime		√
waitInterval		√
dropToProcess	√	

Table 10.1: Load attributes

in the AutoMod software, thus they cause events to happen, and executes logic. The logic in AutoMod, is written as instructions for the loads to follow, while traveling through the system.

After loads are created, they move logically from one process to another, executing actions for the process, which are contained in arriving procedures. When loads are ready to leave the system, they are sent to die, meaning they disappear from the simulation. Each load has a user-defined description called a load type.

All loads have the same attributes in AutoMod, meaning that even though we have different load types with distinct properties, every load in the AutoMod model have the same attributes. Table 10.1 lists the load attributes. The load type L_dummy is used to control a vehicle and to generate time events(see section 14.1.3), and the L_block is used the represent a ship block in the system.

Every physical element that should be visualized requires some graphics to represent the element in the simulation. We have designed a general graphics cube in ACE to represent a ship block with the following dimensions:

- width = 1 meter
- length = 1 meter

- height = 1 meter

When loads are created in the system, we scale the load graphics according to its real dimensions, which are read from a database. Figure 10.2 shows how the block frame is designed in ACE, figure 10.3 shows the general ship block in color.

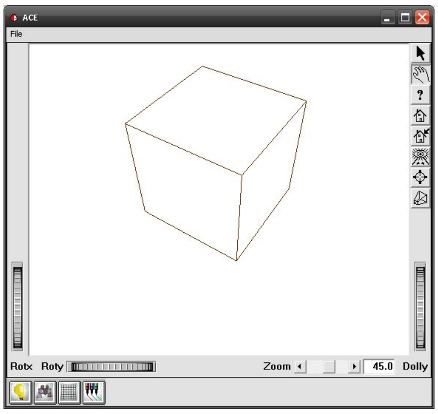


Figure 10.2: Design of a ship block frame in ACE

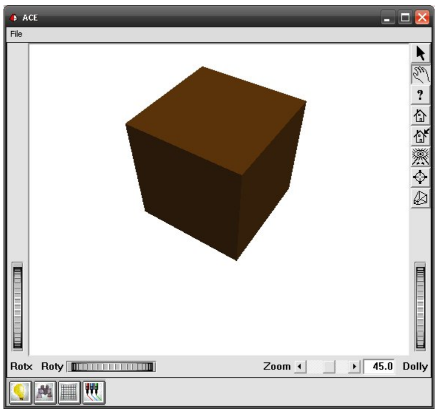
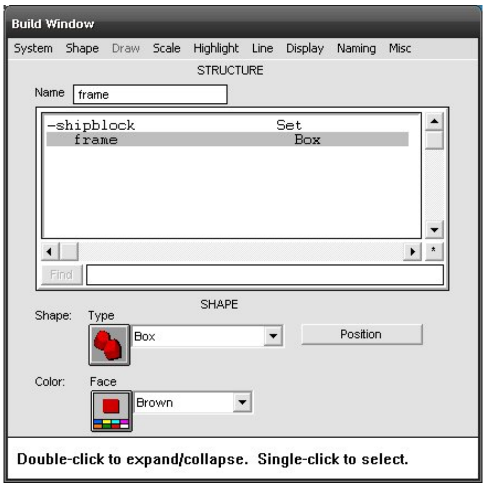


Figure 10.3: The ship block in color

The build windows in ACE, shows which ACE elements compose the graphics, as seen in figure 10.4 we have a set with a box, and the dimension is set to 1x1x1 m^3.



STRUCTURE

Name: frame

```
-shipblock          Set
   frame              Box
```

SHAPE

Shape: Type: Box

Color: Face: Brown

Double-click to expand/collapse. Single-click to select.

Figure 10.4: ACE Build Window for a ship block

10.3.2 Buildings

Designing the graphics of the buildings at the shipyard can be a bit tricky; first of all, graphics in AutoMod are designed with the program ACE. ACE is a graphic editor provided with AutoMod for representing entities in an AutoMod model. Graphics for entities such as vehicles, loads, kinematic machines, buildings and all sorts of elements can be designed with ACE.

Buildings are permanent elements in the simulation, which are often used to produce ship blocks, equip ship blocks, or to paint them, thus we can use an AutoMod queue to represent a building. The graphics for every building are designed with ACE, by using multiple sets, we can make things like the building walls, the roof and windows. If we look at figure 10.5, 10.6 and 10.7 we see how the building called "HAL SYD" have been designed. Figure 10.7 shows the two sets "HAL SYD" is composed of. Figure 10.5 shows the design of the building called "HAL SYD" at OSS.

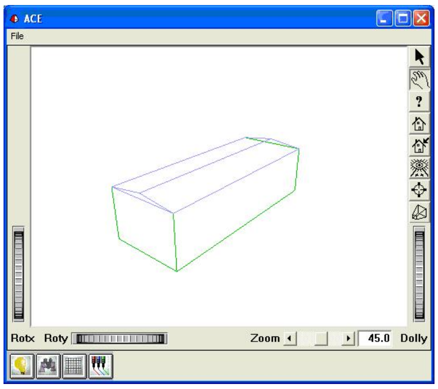


Figure 10.5: Design of Building Frame

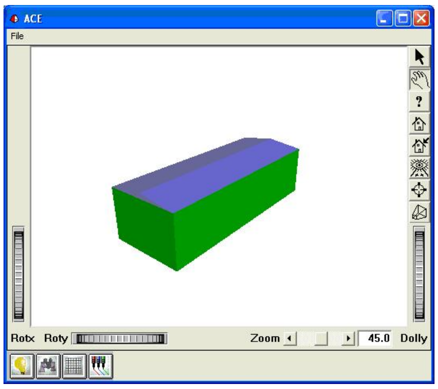
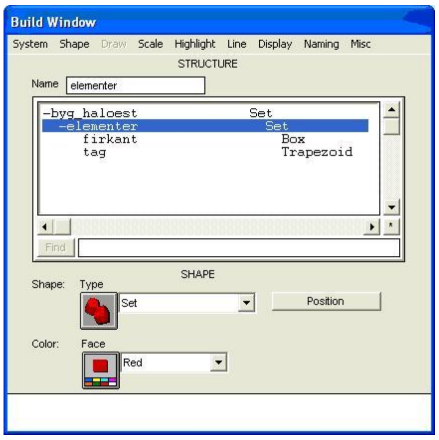


Figure 10.6: Design of Building in Color

As seen from the build window in figure 10.7, graphics of a building consists of multiple elements; a set can contain multiple objects, such as other sets and actually visible objects such as a box and a trapezoid, which are two elements we have generally used for designing the buildings at the shipyard.

By using aerial photographs we are able to estimate the heights of the various buildings, so that our model of the shipyard appears similarly to reality.

Example of an aerial photograph of OSS at figure 10.8 and the corresponding simulation design at figure 10.9.



Name: elementer

```
-byg_haloest            Set
  -elementer              Set
     firkant                Box
     tag                    Trapezoid
```

Shape: Type Set

Color: Face Red

Figure 10.7: Build Window in ACE

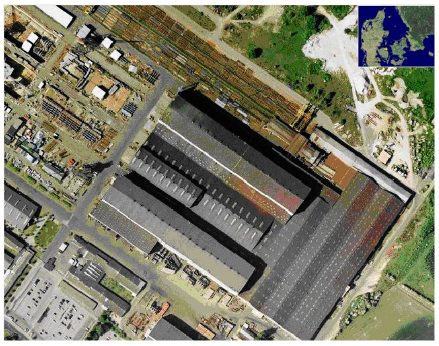


Figure 10.8: Aerial Photograph of OSS

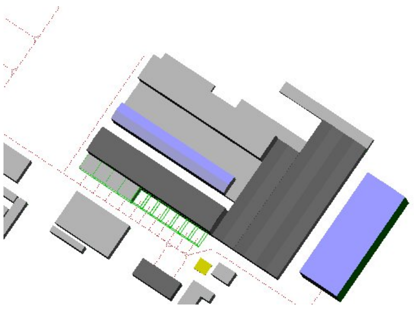


Figure 10.9: Simulation Design of OSS

10.3.3 The road system

In order to model the road system at OSS in the simulation, we can use the AutoMod path movement system. The movement system includes different components, we can choose to simulate the transportation of loads, such as:

- Conveyor Systems
- Power & Free Systems
- Tanks & Pipes
- Path Mover Systems

We have chosen the path mover system, which we find most suitable, since the path simulates the road system at which vehicles can travel, like KA-

MAG vehicles driving at the road system at OSS.

When we have placed all the queues (see section G.8), we construct the path mover system, at which the KAMAG vehicles will transport the ship blocks. Again we create a new system, as with the static system, but this time we choose "Path Mover" instead and we shall name this system "pm". Start by drawing lines as in figure 10.11 using the "Single Line" from the path mover menu at figure 10.10. After designing the road system the lines should be connected, for which we can use "Fillet" from the path mover menu. When finished connecting all lines in the system, we place the control points. Control points determine from where vehicles can hold, thus vehicles basically travel from one control point to another. A good idea is to place control points for every road intersection, giving the vehicle ultimate control to decide where to drive from and to. We have also placed control points at the center of every queue, so the vehicle can stop at the queue center, to drop and pick up loads.

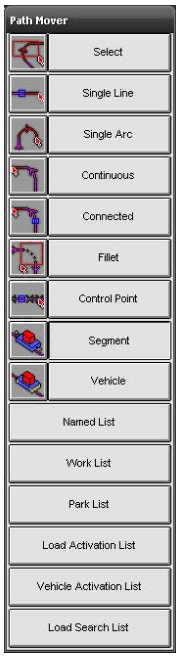
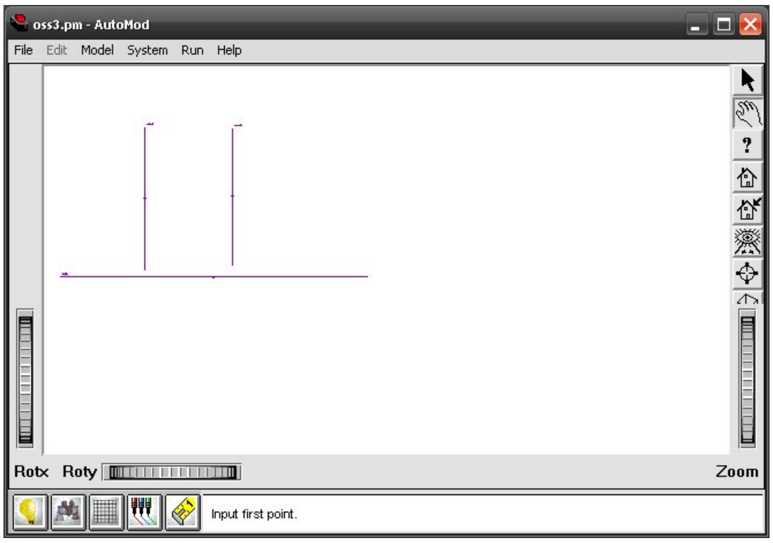


Figure 10.10: Path Mover Menu



Input first point.

Figure 10.11: Draw the roads as lines

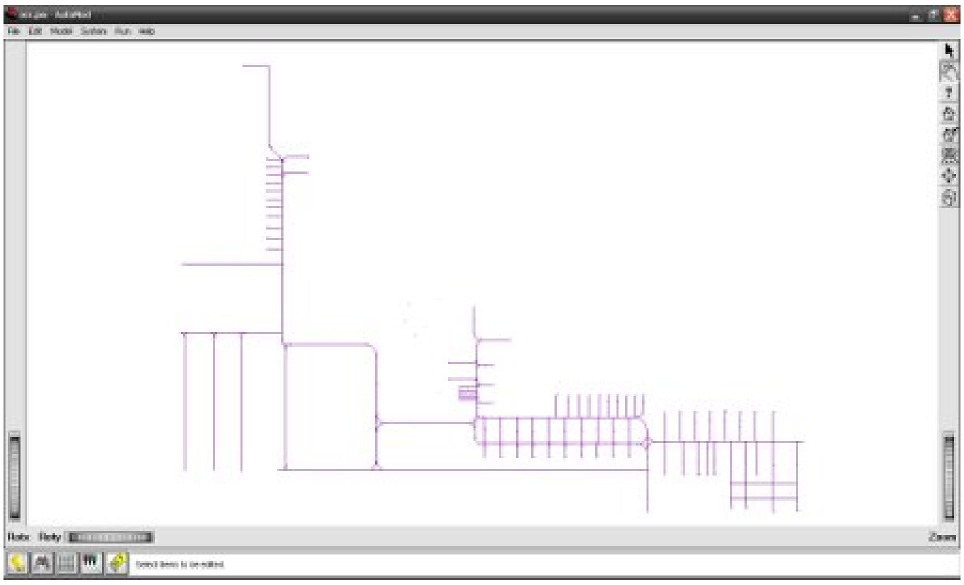


Figure 10.12: Road system design of OSS

10.4 Modeling the virtual control

The logic in AutoMod is placed in processes and functions. Processes are used as the virtual control of the load flow in the system. For every queue in the system we have a process handling the flow of that specific queue. Also KAMAG vehicles have individual processes that control the vehicle behavior. AutoMod have been designed for production engineers, thus normally we are supposed to write logic, that handles a load flow from start to end in the system. But in our case, we do not know the system flow of the loads, because we are designing an external system that is controlling the flow in the system.

In this section we describe the processes that control queues and vehicles, since they are essential in the system. We will also explain the processes and functions used to communicate with and control the simulation model externally.

10.4.1 Controlling KAMAG vehicles

Figure 10.13 shows the flow of a load in the process of KAMAG vehicle 4843, but the process is similar for the other KAMAG vehicles. We have two types of loads in the system; a dummy load "L_dummy" and a ship block load "L_block". Dummy loads are placed on KAMAG vehicles when they are instructed to drive to a location in the simulation at runtime, because a vehicle in AutoMod can not travel without carrying a load, because all logic concerns the load, which means that we have to place a dummy load on the vehicle we want to control, and write some logic for this load.

When we want to force a vehicle in a simulation to move, we place a dummy load on it, and set the load attribute "transportNext" to the destination of the vehicle. The vehicle travels to the next location, and the dummy load is send to die, meaning that it is removed from the model (see figure 10.13).

When a KAMAG vehicle is carrying a load it will check the attribute "transportNext"of load "L_block" to see if it needs to travel. If the transportNext attribute is set, we check to see if the next location equals the KAMAG vehicles current location, if this is not the case, the vehicle will travel to the next location and call a procedure "F_arrivedToDestination" that will generate an event notifying the external system that the vehicle has reached its destination, else we check to see if the load attribute "putDown" is set, if this is the case, the KAMAG vehicle is in the queue, where it should place the load. We then generate an event regarding placing the load in the queue, and the load is send to the process, which controls the queue and the load is visually placed in the queue.

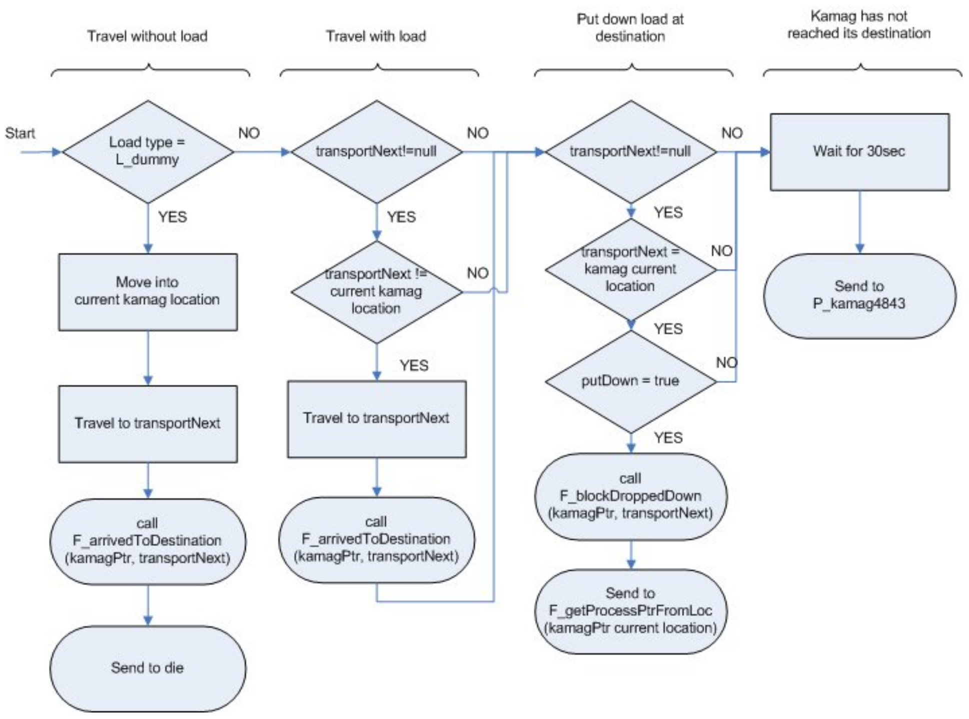

Travel without load
Travel with load
Put down load at destination
Kamag has not reached its destination
Start
Load type = L_dummy
NO
YES
Move into current kamag location
Travel to transportNext
call F_arrivedToDestination (kamagPtr, transportNext)
Send to die
transportNext!=null
NO
YES
transportNext != current kamag location
NO
YES
Travel to transportNext
call F_arrivedToDestination (kamagPtr, transportNext)
transportNext!=null
NO
YES
transportNext = kamag current location
NO
YES
putDown = true
NO
YES
call F_blockDroppedDown (kamagPtr, transportNext)
Send to F_getProcessPtrFromLoc (kamagPtr current location)
Wait for 30sec
Send to P_kamag4843

Figure 10.13: Process P_kamag4843

10.4.2 Controlling queues

When a load is placed in a queue in AutoMod it will not automatically stay in that queue unless we instruct it to, which makes it necessary constantly to send the load back in the queue as seen in figure 10.14. Every queue has a process that contains the queue logic. When a load is sendt to a queue process, the process will place the load in its corresponding queue. Unless a vehicle arrives at the queue, and the queue has been instructed, that it should be picked up by that vehicle, the process will send the load to itself. In the other case, if a vehicle has arrived at the queue to pick up the load, the load will be send to a KAMAG manager process, which will forward the load to the vehicle residen in the queue.

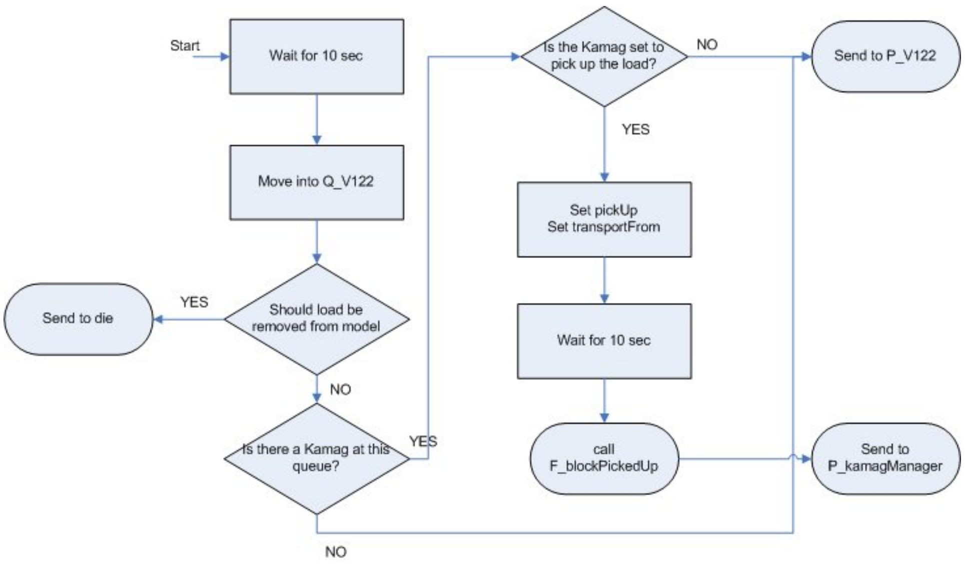


Figure 10.14: Process P_V122

10.4.3 External access to simulation model

In order to communicate with and control the simulation from an external system, it is necessary design and implement processes and functions for this purpose. It has been quite difficult and comprehensive to get access to get generic access to all our processes and functions from the outside of the simulation model, thus this task required several hacks in the coding. In order to get hold on the various pointers in the simulation model we have designed the following functions:

- **F_getQueuePtrFromLoc :** This function takes a location pointer as input, and returns the queue, wherein the location resides.
- **F_getProcessPtrFromLoc :** This function takes a location pointer as input and returns the corresponding process pointer, which means the process controlling the queue, wherein the location resides.

It is not possible from within AutoMod to get in hold of a pointer, simply by knowing its name, i.e. the pointer VPPtr_V122, which is pointing at the process P_V122, is not accessible from AutoMod simply by knowing the name "VPPtr_V122" from AutoMod, but instead it is odd enough accessible from the outside of AutoMod, meaning the ActiveX object created in the Middleware, thus when we need to get in touch with a pointer to a location, queue or process, it is necessary to go trough the middleware strangely enough.

As seen in figure 10.13, we call the function F_getProcessPtrFromLoc from the corresponding KAMAG process, when a KAMAG has reached its destination(control point) and must place the load it is carrying at a storage location, because the KAMAG process needs to know which process controls that storage location, so the load can be send to the correct next process. The function F_getProcessPtrFromLoc is called with a control point as argument, and returns a pointer to the process controlling the queue, which is placed at the control point. Yes, it indeed seems messy, but since AutoMod do not provide data structures like hashtables to retrieve pointer from string names (mapping), this is the approach we have decided to use.

10.5 Placing the physical elements

We have now designed the different elements in our simulation model, and it is now time to unite all the elements, such as queues, buildings, ship blocks, KAMAG vehicles, and virtual controlling elements, into one complete AutoMod simulation model. With AutoMod we can scale every building after real world scale, thus we have received an dxf file from OSS with a complete overview of the shipyard, including every storage location, building and the road system. Figure 10.15 shows the CAD file opened with AutoCad, where the red lines indicate storage locations, the gray areas with grey dot-and-dash lines indicate buildings, and the roads are black(no color)limited at both sides.

AutoMod can read this file as a background image, we then started by placing each queue (buildings and storage locations), and scaling the queue[2] to

[2]See appendix G.8

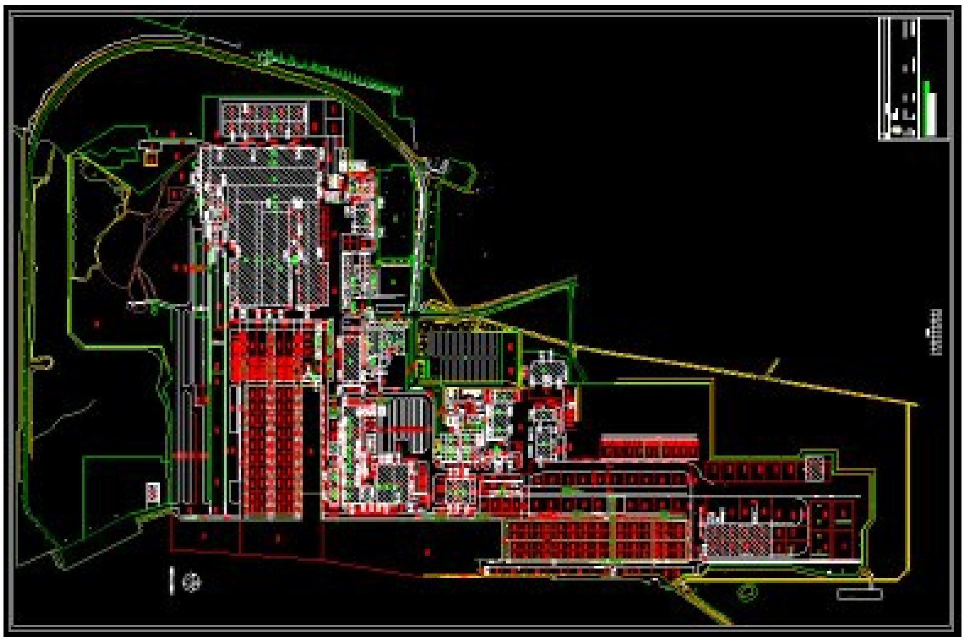
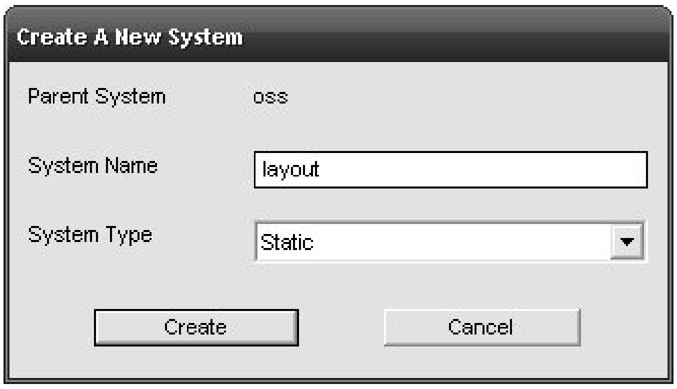


Figure 10.15: Autocad file gives overview of OSS

it fits precisely, thereby making a simulation model, which is a precise measure of the real world shipyard.

We create a new system, by selecting "System" in the AutoMod file system, then select "New" and select "static" system as seen in figure 10.16, and give the system a name, i.e. "layout".



Create A New System

Parent System: oss

System Name: layout

System Type: Static

Figure 10.16: Create a new system

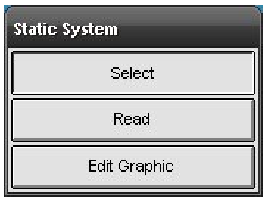
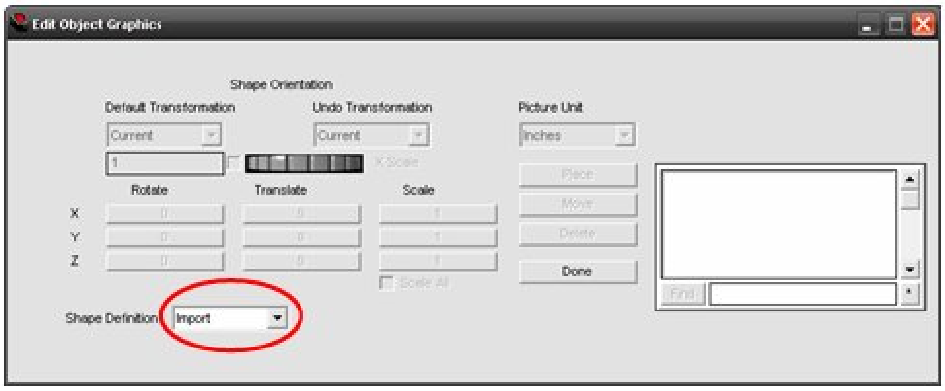


Figure 10.17: Menu in a static system

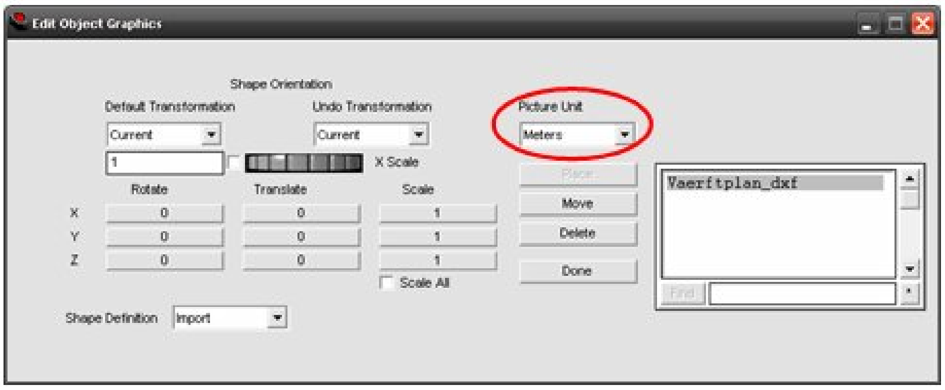
A new menu will appear after creating a static system, as seen in figure 10.17, choose "Edit Graphic", and the choose "import" as seen in figure 10.18, and AutoMod will start to import the file as a static system.



Default Transformation: Current
Undo Transformation: Current
Picture Unit: Inches
Shape Definition: Import

Figure 10.18: Edit Object Graphics

After the file has been imported, choose the measure system as seen in figure 10.19. You can translate, rotate and scale the image as you prefer. We do not need to scale our image, because OSS already uses the correct scaling values in the image.



Default Transformation: Current
Undo Transformation: Current
Picture Unit: Meters
Shape Definition: Import
Vaerftplan_dxf

	Rotate	Translate	Scale
X	0	0	1
Y	0	0	1
Z	0	0	1

Figure 10.19: Choosing picture unit

Now we have a static system with OSS as a background as seen in figure 10.20. We will place a queue for every storage location, and for every building. Every queue will have the same graphics with only the scaling varying to fit the static system. Each building has been given an individual design see section 10.3.2.

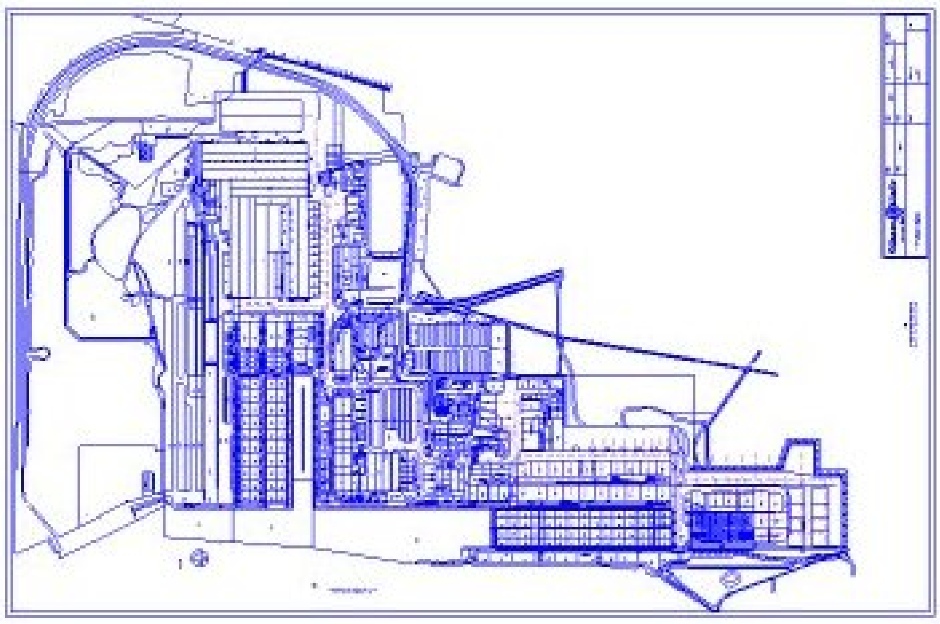


Figure 10.20: Static System with OSS

10.6 Communication

In order to control the simulation model, it is necessary to implement some form of communication in the simulation model. There are two possibilities to communicate with AutoMod, which are sockets or ActiveX. As described earlier we have chosen ActiveX as communication interface between the simulation model and the middleware.

The AutoMod simulation environment can be accessed with the use of ActiveX. This is done via the AutoMod runtime object which can be used as an ordinary object in the programming languages which support ActiveX, as for example Visual Basic or C#. An overview of the ActiveX component in AutoMod can be seen in figure 10.21.

The methods, properties, events and their description can be seen in table 10.2,10.3 and 10.4 respectively.

The syntax for each method, with parameters and return values can be seen in appendix H.

Method	Description
CallFunction	Calls a user-defined AutoMod function in the model during a simulation.
GetVariable	Gets the current value of an AutoMod variable in the simulation.
SetVariable	Sets the current value of an AutoMod variable in the simulation.
OpenModel	Opens an AutoMod simulation model, which is an AutoMod compiled .exe file.
CloseModel	Closes the opened simulation model
DisplayView	Changes the view in the simulation
OpenLogFile	Create a diagnostic file for debugging. The log file contains diagnostic information that is generated during the simulation; it is useful for debugging errors associated with the custom interface.

Table 10.2: AutoMod ActiveX methods and their description

Property	Read Only	Description
Animating	No	Turns model animation on or off.
CurrentClock	Yes	Gets the current value of the simulation clock.
DisplayStep	No	Changes the simulation's animation step (that is, the length of the interval between animation updates
Paused	No	Pauses or continues the simulation.
State	Yes	Determines the current state of the simulation.

Table 10.3: AutoMod ActiveX properties and their description

Events	Description
OnModelReady	Used to determine when the simulation is loaded and ready to run.
OnStateChange	Used to determine when state changes occur in the simulation (for example, when the simulation is paused, starts running, or completes).
OnUserEvent	Used to send information from AutoMod to the middleware. The event is generated whenever the FireUserEvent function is called in the AutoMod simulation.

Table 10.4: AutoMod ActiveX events and their description

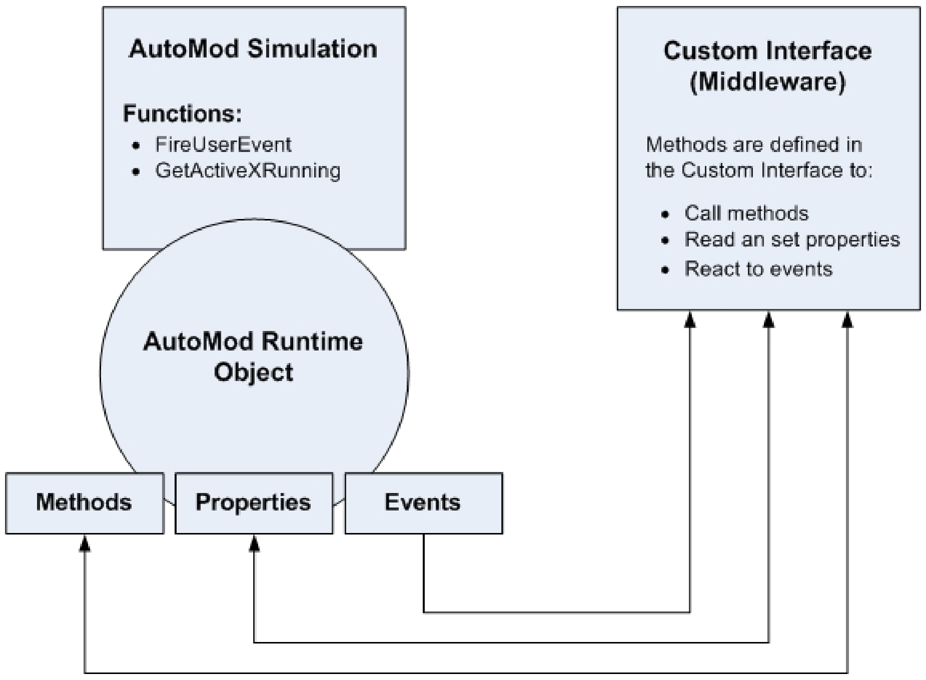


Figure 10.21: ActiveX Overview

Chapter 11

DECAF

In this chapter we will design the Multi-Agent System by the use of the DECAF framework. With this design will try to fulfill the requirements in section 2.4.

We will first identify the agents that are needed. Afterwards we will define their capabilities in the form of which actions they can perform to solve a particular task. The agents capabilities are defined with the help of the DECAF graphical plan editor.

11.1 Agents

From the constraints and overall requirements we identify following agents, which we will model in DECAF:

- **C-planners**
- **D-planner**
- **KAMAG vehicles**

Besides these agents we identify following agents:

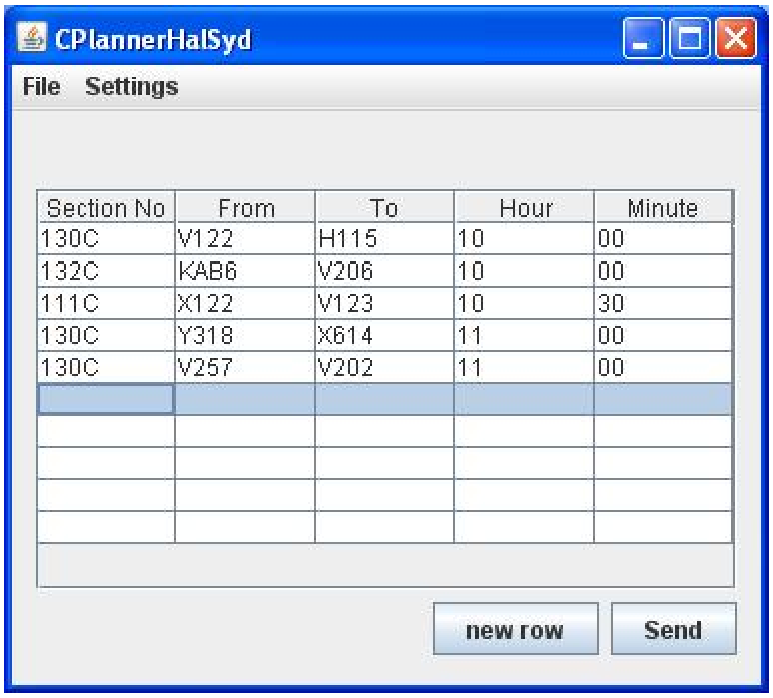
- **Init Agent:** Initializes the multi-agent system and the simulation model, by placing ship blocks and KAMAG vehicles at initial locations at system start.
- **Communication Agent:** Handles all agent requests to the simulation model, as well as simulation responses to the multi-agent system.

11.2 Agent Capabilities

We will now discuss each agent with it's capabilities.

11.2.1 C-planner

The C-planner agent consists of an Input GUI, where the C-planner can enter the transportations he need during the day. This GUI is shown in figure 11.1. The C-planner can request transportations from the D-planner, which then coordinates the transportations and makes a daily transportation plan. When the D-planner has coordinated the requests, each C-planner will get the part of coordinated plan with their requests back. These responses from the D-planner can be seen in the Monitor GUI. If a C-planner is unhappy with the coordinated request, he will be able to make a modification request by the use of the Monitor GUI.



Section No	From	To	Hour	Minute
130C	V122	H115	10	00
132C	KAB6	V206	10	00
111C	X122	V123	10	30
130C	Y318	X614	11	00
130C	V257	V202	11	00

Figure 11.1: C-planner Input GUI

The tasks that the C-planner agent can perform can be seen in figure 11.2. A description of each task is given in table 11.1.

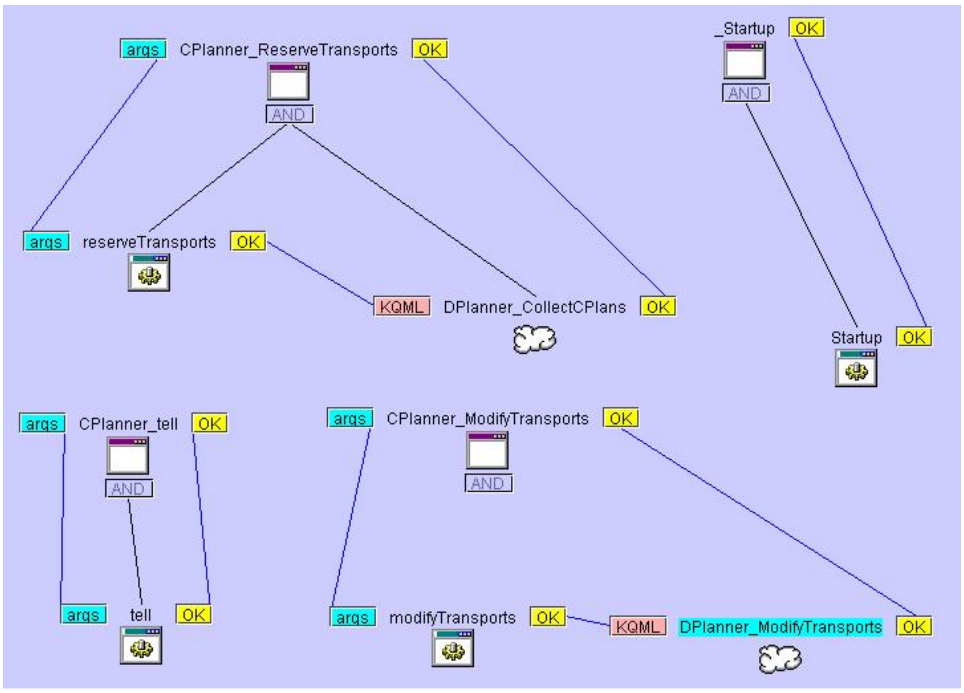


Figure 11.2: The Tasks and Actions that the C-planner can achieve

Task	**Description**
_Startup	The _Startup task is a standard task in the DECAF architecture, which is run only once, that is when the agent starts [27, section 5.2.1]. For the C-planner Agent this task will create an Input GUI like the one shown in figure 11.1 and a Monitor GUI.
CPlanner_ReserveTransports	The ReserveTransports task is run, whenever the C-planner presses on the send button in the input GUI. This task will then receive the requested transportations from the input GUI, which is placed in the parameter args (see the green box in figure 11.2). The reserveTransports action will then create a KQML message with the appropriate syntax whereupon the request message will be sent to the D-planner.
CPlanner_tell	This task is run whenever the D-planner responds back to the C-planner requests. This task will update the Monitor GUI of the C-planner.
CPlanner_ModifyTransports	If the C-planner is unhappy with the coordinated transportation requests, then he is able to ask for modifications from the D-planner with this task.

Table 11.1: C-planner Tasks

11.2.2 D-Planner

The D-planner receives transportation requests from the C-planners. Coordinates these requests into a daily transportation plan, in co-operation with the KAMAG agents, by asking them about their drive plans and state. To keep it simple, we have aggreed that the KAMAG agents can only handle one transportation request at a time, that is a KAMAG can accept to drive a transportation task and set its state to busy. After completion of the transportation task, the KAMAG will set its state to free and accept another transportation task from the D-planner.

The D-planner coordinates the transportation requests by ordering the requests according to the starttime of the transportations. If two or more transportation tasks have the same starttime, then the D-planner will sort these tasks by their travel distance.The plan coordination will happen periodically for every X minutes, that is the D-planner will be triggered for every X minutes and calculate another plan, if there has come newer requests since last trigger. The flow diagram for the coordination of the transportation tasks can be seen in figure 11.3

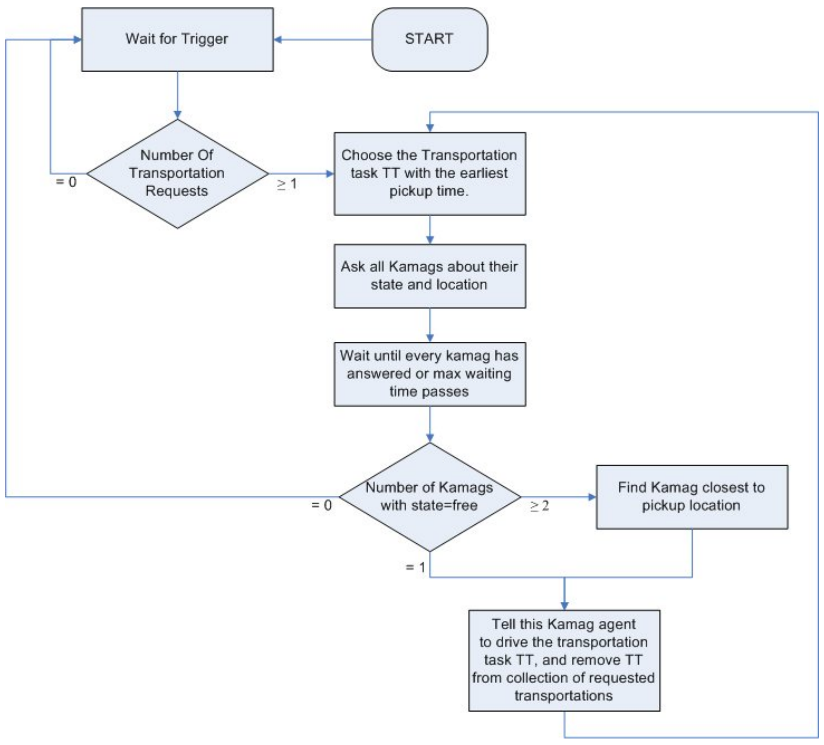

Wait for Trigger
START
Number Of Transportation Requests
= 0
≥ 1
Choose the Transportation task TT with the earliest pickup time.
Ask all Kamags about their state and location
Wait until every kamag has answered or max waiting time passes
Number of Kamags with state=free
= 0
≥ 2
= 1
Find Kamag closest to pickup location
Tell this Kamag agent to drive the transportation task TT, and remove TT from collection of requested transportations

Figure 11.3: D-planner Coordination of Transportation Tasks

The tasks that the D-planner can execute can be seen in figure 11.5. The description of the tasks can be seen in table 11.2. The last four tasks in this table are used for coordination of the transportation tasks (flow diagram can be seen in figure 11.3) and the task DPlanner_CollectCPlans is used by the D-planner to collect C-planner transportation requests (flow diagram can be seen in figure 11.4).

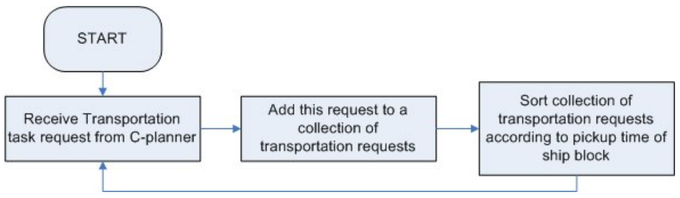

START
Receive Transportation task request from C-planner
Add this request to a collection of transportation requests
Sort collection of transportation requests according to pickup time of ship block

Figure 11.4: Flow diagram for collection of transportation request

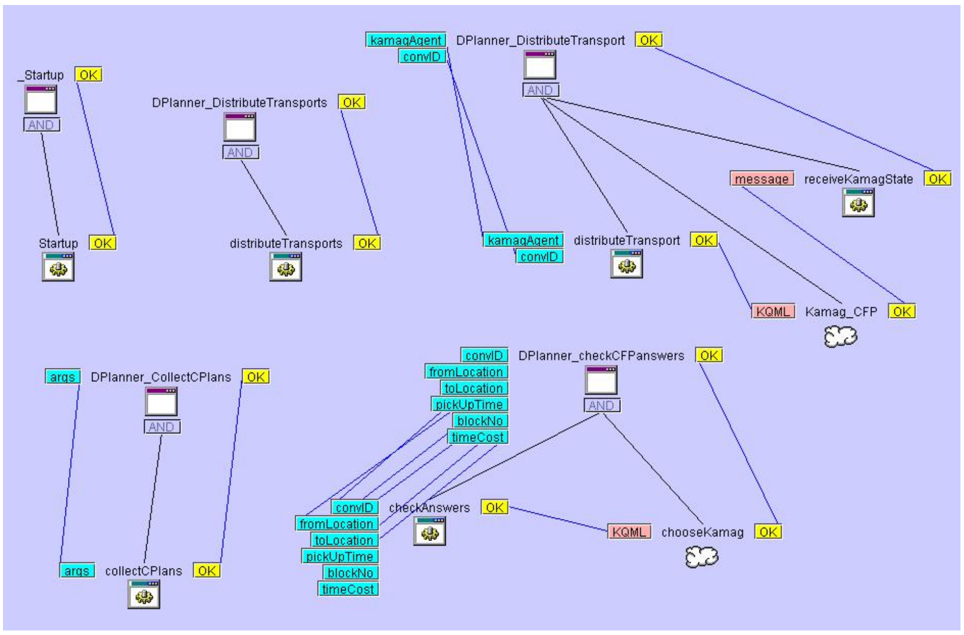


Figure 11.5: The Tasks and Actions that the D-planner can achieve

11.2.3 KAMAG

The KAMAG agents are responsible to transport ship blocks from one location to another. The KAMAG agents will be asked about their state and location whenever there is a transportation request to be fulfilled. They will

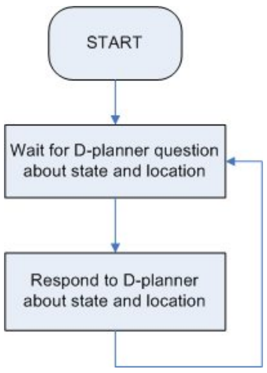
Task	Description
_Startup	
DPlanner_CollectCPlans	This task is used to collect transportation requests from the C-Planners. Whenever a C-planner makes a transportation request, this task will create a "transporttask object", with information about the request. The "transporttask object" will be placed in a collection of requests to be handled by the D-Planner. The collection of requests are sorted first by the pickuptime; if there are equal pickup times then sorted by transport distance.
DPlanner_DistributeTransports	This task is called periodically and checks whether there are unhandled transportation requests. If so, this task will use the next two tasks in this table for; 1. send KQML messages to all KAMAG agents, asking their state (free/busy) and location 2. Check answers from KAMAG agents about their state and location, then choose the best suited KAMAG for the transportation task.
DPlanner_DistributeTransport	This task is run by the DPlanner_DistributeTransports task to send one KQML message to a specific KAMAG, asking for its state and location.
DPlanner_CheckCFPanswers	This task is also run by the DPlanner_DistributeTransports task and has the responsibility of checking all responses from the KAMAG agents about their state and location, and choose the best suited KAMAG for the transportation task
DPlanner_tell	Whenever a KAMAG receives a request about its state and location, it will send these informations to the D-planner. The D-planner will receive this information with this task.

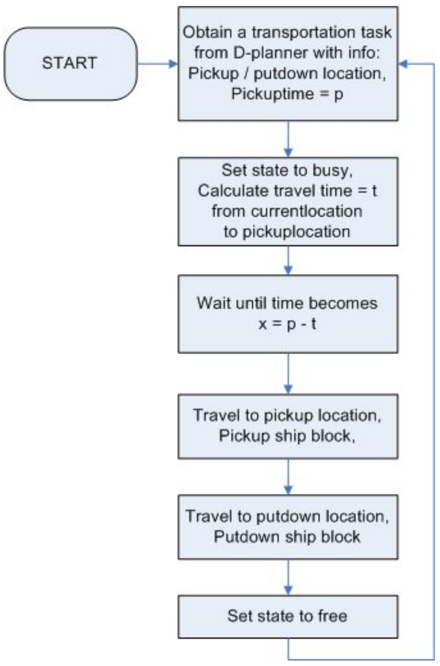
Table 11.2: D-planner Tasks

answer the D-planner, whereupon the best suited KAMAG will get a response from the D-planner to fulfill the transportation request, with information about where to pickup/putdown the ship block, and what time to pick it up(pickuptime **p**). The chosen KAMAG agent will then sets its state to busy, calculate the travel time **t** from its current location to the pickup location and wait. When the time becomes **p-t**, the KAMAG agent will drive to pick the ship block up, transport it to the destination location, put it down and set its state to free.
So a KAMAG agent has two tasks

1. Respond, when asked, to the D-planner about the current state and location
2. Receive a transportation task and fulfill it.

The flow of these can be seen in figure 11.6 and 11.7.


START
Wait for D-planner question about state and location
Respond to D-planner about state and location
Figure 11.6: KAMAG task flow 1


START
Obtain a transportation task from D-planner with info: Pickup / putdown location, Pickuptime = p
Set state to busy, Calculate travel time = t from currentlocation to pickuplocation
Wait until time becomes x = p - t
Travel to pickup location, Pickup ship block,
Travel to putdown location, Putdown ship block
Set state to free
Figure 11.7: KAMAG task flow 2

11.2.4 Init Agent

The init agent consists of a GUI to input initial placements of ship blocks and KAMAG agents at the OSS into the multi-agent system and simulation when the system starts. The Init agent has only two tasks which can be seen in figure 11.9. A description of these tasks are given in table 11.3

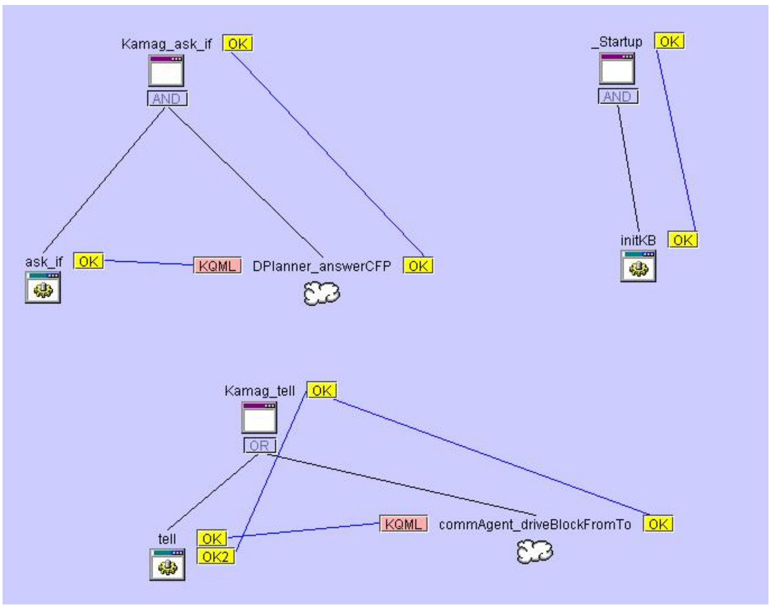


Figure 11.8: The Tasks and Actions that the KAMAG can achieve

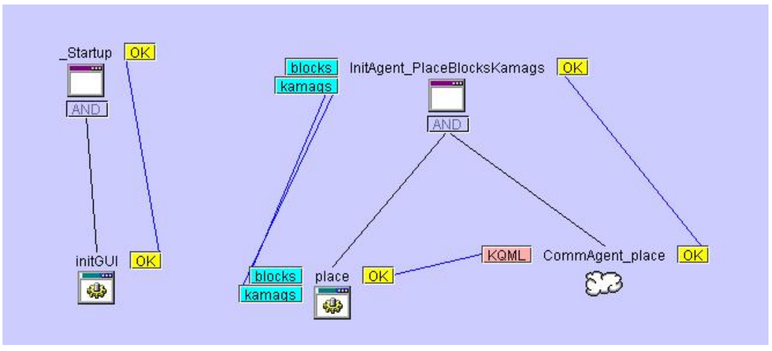


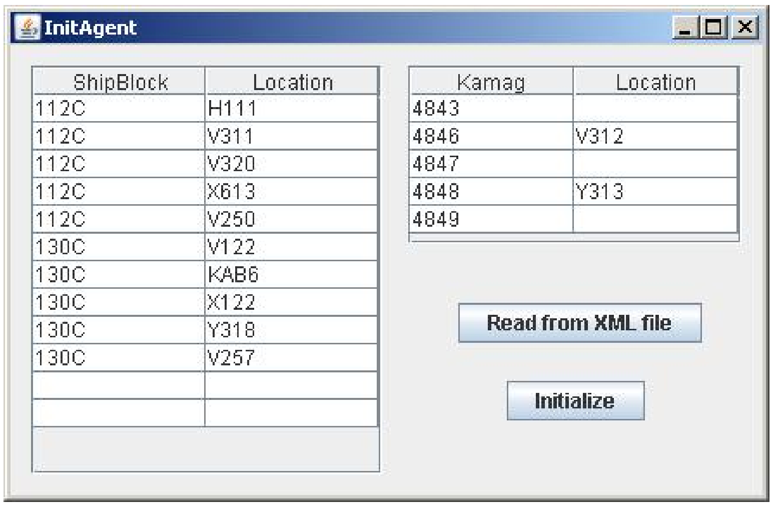
Figure 11.9: The Tasks and Actions that the Init Agent can achieve

11.2.5 Communication Agent

The Communication agent handles all agent requests to the simulation model, as well as simulation responses to the multi-agent system. The tasks and actions that the communication agent can achieve can be seen in figure 11.11. A description of these tasks are given in table 11.4 (the first part of the task names “CommAgent” are not shown in the table to give room for the description field). The type column indicates:

Task	Description
_Startup	This task initializes the input GUI, which can be seen in figure 11.10
InitAgent_PlaceBlocksKamags	This task is used by the input GUI to insert data of the initial placements of the ship blocks and KAMAG vehicles into the multi-agent system and simulation.

Table 11.3: Init Agent Tasks



ShipBlock	Location
112C	H111
112C	V311
112C	V320
112C	X613
112C	V250
130C	V122
130C	KAB6
130C	X122
130C	Y318
130C	V257

Kamag	Location
4843	
4846	V312
4847	
4848	Y313
4849	

Figure 11.10: The GUI for the init agent

- req = This task **request**'s to make an execution happen in the simulation
- res = This task handles a **response** from the simulation, because there has occured a specific event in the simulation.

Task	Type	Description
_Startup		Initializes a collection that will contain times and each time will be connected with one or more agents. For each of the times the simulation will respond back to the connected agents, when the time has been reached.
_DriveToLocation	req	This task can be achieved by a KAMAG agent to drive to a location in the simulation.
_PickUpBlock	req	This task is achieved by a KAMAG agent to pick up a ship block in the simulation
_PutDownBlock	req	This task is achieved by a KAMAG agent to put down a ship block in the simulation
_AddRequestTimeEvent	req	This task is achieved by a KAMAG agent to make the simulation notify the KAMAG agent when a specific time has been reached in the simulation
_PlaceBlocksKamags	req	This task is achieved by the Init Agent to initialize the simulation model by placing ship blocks and KAMAG vehicles to initial locations in the simulation.
_ArrivedToDestination	res	This task notifies a KAMAG agent, when the KAMAG has arrived to a destination in the simulation
_BlockPickedUp	res	This task notifies a KAMAG agent, when the KAMAG has picked up a ship block in the simulation
_BlockDroppedDown	res	This task notifies a KAMAG agent, when the KAMAG has dropped down a ship block in the simulation
_TimeEventGenerated	res	This task notifies an agent, which has requested to be notified when a specific time has been reached in the simulation

Table 11.4: Communication Agent Tasks

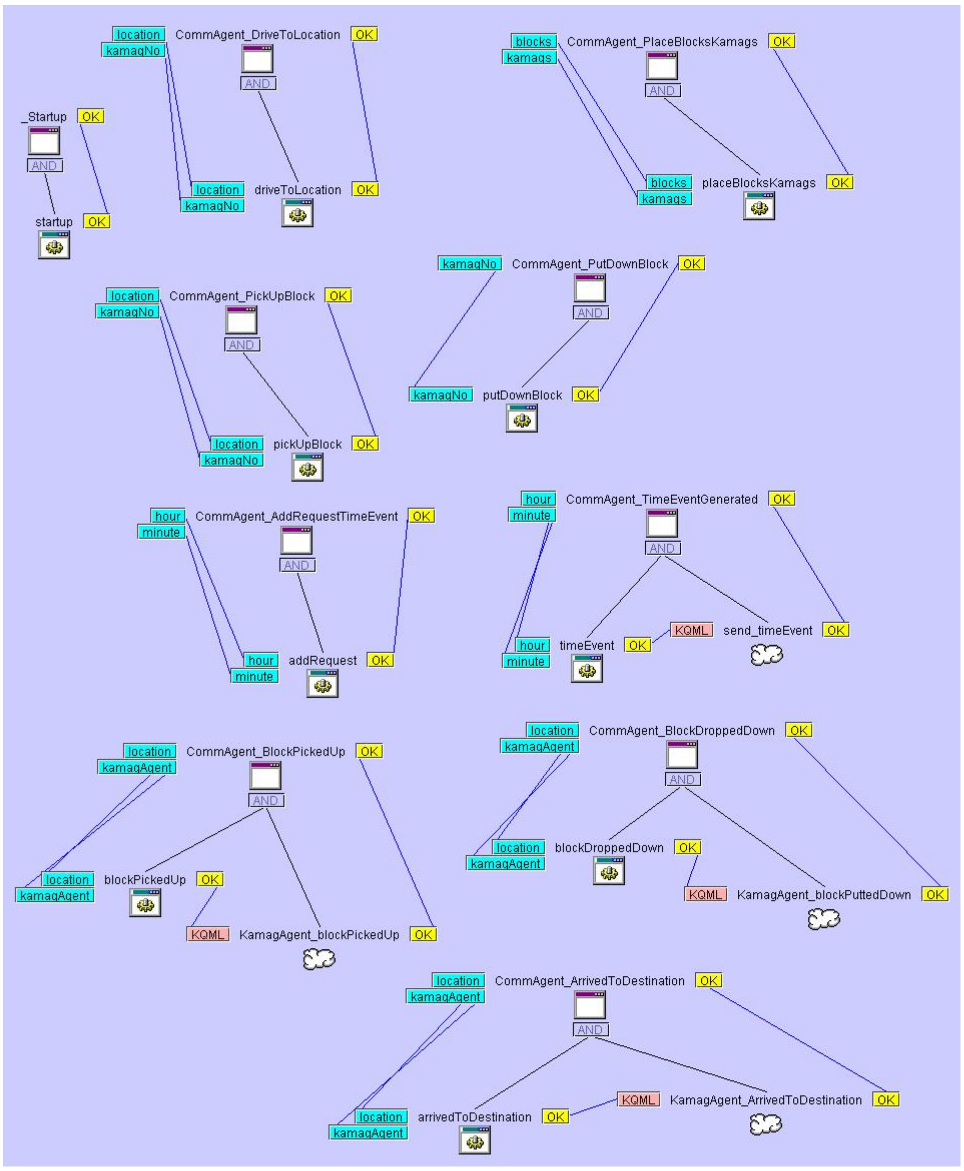


Figure 11.11: The Tasks and Actions that the Communication Agent can achieve

Chapter 12

Cougaar

This chapter briefly presents the Cougaar framework and the Cougaar methology, from where the multi-agents system have been designed. In this chapter we will use the "Cougaar Design Methodology" [20], to map the OSS domain and the set of business processes onto the Cougaar concept. The workflow can be seen in figure 12.2.

12.1 A brief overview of the Cougaar framework

The agent capabilities and behaviours are defined by the plugins in the agent [20]. Organized agents are called communities in Cougaar, meaning that agents that have a common functional purpose can be grouped into communities. In general, agents that reside in the same community will have the same role, and some agents might be associated with more than one community, while other agents may not be related to any community. A society in Cougaar is a collection of agents that must interact in order to solve an overall common objective. The collection of all communities/agents is referred to as the society. The behavior of a Cougaar society is the aggregate emergent behavior of all the agents in the society cooperatively working on a set of requirements.

The Cougaar framework is based on a publish/subscribe pattern, where every plugin in an agent communicates by publishing and subscribing objects to a common blackboard. Every agent has its own blackboard whereto it can publish and subscribe information. A plugin has no knowledge regarding who subscribes/publishes what.

When agents are organized into communities, communities must have a relationship in order to interact. There are two common types of relationships, which are predefined relationships in a so called organizational assets, that are either based in the customer/provider relationship or the superior/subordinate. When dealing with a system that typical processes a few long term tasks which spans over several days, week or months the superior/subordinate relationship is preferred, while the customer/provider relationship is preferred when dealing with many short term tasks like transport tasks for example. An example of a Cougaar agent is seen in figure 12.1.

The interacting between agents in different societies are conducted using so called PlanElements, which contain a task

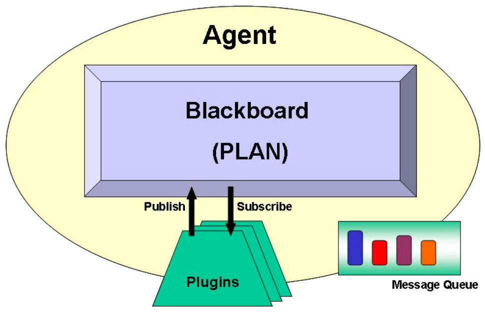


Figure 12.1: Cougaar Agents is composed of plugins. The Plugin subscribes/publishes to a common agent blackboard

12.2 Cougaar Methology

The Cougaar framework proposes a methology for designing a multi-agent system with Cougaar. We intend to follow this methology which can seen in figure 12.2.

12.3 Agent Enumeration

In the transportation problem at OSS, we have a number of choices for breaking the problem down into Agents, as discussed in section 9.1. There is one company: we could model everything in one Agent. There are in fact several players, but it seems that there are three fundamental groups who play a role here: planners (C-planners), schedule coordinators (D-planners, usually only one at a time) and Kamag vehicle drivers. They have their own assets, such as people(workers), machines and vehicles and their own business processes, and by joining them into one agent we would lose the natural decomposition, and by breaking them down further we would increase network traffic to coorporate among the groups.

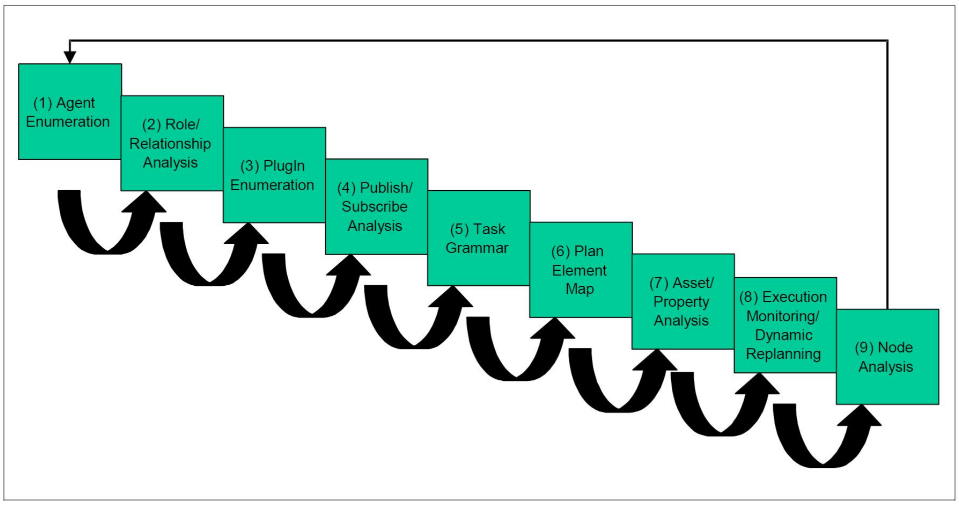

(1) Agent Enumeration
(2) Role/ Relationship Analysis
(3) PlugIn Enumeration
(4) Publish/ Subscribe Analysis
(5) Task Grammar
(6) Plan Element Map
(7) Asset/ Property Analysis
(8) Execution Monitoring/ Dynamic Replanning
(9) Node Analysis

Figure 12.2: Cougaar Design Methodology Workflow

The agents in the system can be determined by the fundamental actors, whose behaviors and interactions we wish to model in the system, which include the following at OSS:

- **C-planners:** request several transportations from the D-planner on daily basis.
- **D-planner:** generates an overall transportation schedule and provides this to the C-planners. Delegates transportation tasks to kamag vehicle drivers.
- **Kamag vehicle drivers:** transports ship blocks between locations

Which leads to the following agents in the system, as seen in figure 12.3:

- **Planner Agents:** Generates transportation requests and sends them to Coordinators.
- **Coordinator Agents:** Receives transportation requests from planners, coordinates theese transportation requests into an overall schedule and delegates transportation tasks to Kamag Vehicles. Then sends updated schedules back to planners.
- **Kamag Vehicle Agents:** Receives transportation task, gives etimates on task completion time and reports when tasks are completed.

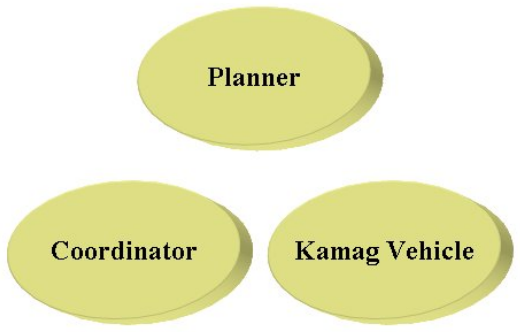


Figure 12.3: Agent enumeration

12.4 Role/Relationship Analysis

In this section we describe the roles and relationships among the agents.

12.4.1 Kamag Vehicle Agent

A KVA provides a transportation service to a coordinator agent, and more specifically to pick up a ship block at a given location on a deadline and deliver the ship block to a specified location. A KVA has a plan of reservations, thus when a request is received it checks if it is possible to perform the task, and reports back. The KVA has a TransportProvider role.

12.4.2 Coordinater Agent

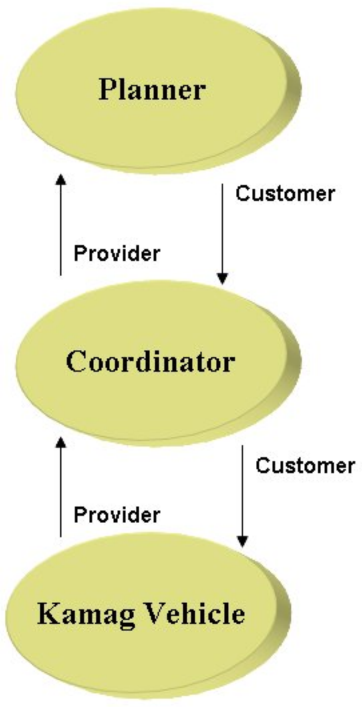
The CA provides a coordination service for planner agents, more specifically it receives transportation reqeuests from PAs, coordinates them and delegates transportations to kamag vehicle agents. The CA has a Transport-Coordinator role.

12.4.3 Planner Agent

The PA do not provide any service to other agents, it subscribes to a coordination service from the coordinater agent, by publishing transportation requests and expecting them to be coordinated and to get a time estimate on when tasks are expected to be completed, and get a report if a request is not possible to satisfy. The PA has a TransportPlanner role.

Agent	Role/Relationship
PA	TransportPlanner/(No services provided: customer of CA)
PA	TransportCoordinator/Provides a task coordination service for PA, and is customer of KVA)
KVA	TransportProvider/Provides a transportation service for CA)

Table 12.1: Role/relationship overview



1. **PA**s are customers of **CA**s, and the **CA**s (typically only one) are the providers to the **PA**s.
2. **CA**s are customers of the **KVA**s, and the **KVA**s are providers to **CA**s.

12.5 Plugin Enumeration

In this section we compose agents into plugins, which together makes the agent behavior. The plugin enumeration is inspired from the real world, mapping human behavior into agents, where humans involve several strategies over and over, to solve a particular problem. These strategies can be categorized as follows [20]:

1. Gathering information(LDM template), which concerns reading new and changed information from external data sources.
2. Delegating (Allocator template), meaning allocating tasks to appropriate resources for final handling or further disposition.
3. Monitoring(Assessor template), concerning assessing the plan for internal consistency and force replanning when necessary.

4. Reporting(UI template), which concerns reporting information back to users of the system.

The plugins are enumerated and described in their cooresponding agents.

Planner Agent

The planner agent delegates transportation requests for further disposition, and monitors the plan for internal consistency and forces replanning when necessary. The Cougaar framework proposes the following plugin types to satisfy those needs; an Allocator plugin for delegating task and an UI plugin for reporting to the user, thus we have the following plugins:

- PlannerGUI: Provides a GUI to the user, wherein a C-planner can define transportation requests and submit those to the system.
- TransportRequestAllocator: subscribes to the transportation requests and allocates those to the CA for further disposition.
- AllocationAssessor: subsribes to TransportRequest Allocations (plan elements), monitoring the status of allocations for consistency an monitors that they are on schedule and publishes Reports.
- MonitorGUI: Subscribes to Reports and visualize the status in a graphical user interface. from TransportRequestAllocatorPlugin and monitors that they are on schedule, and replans if necessary.

Coordinator Agent

The coordinator agent or TransportCoordinator, receives transport requests from the TransportPlanner, which it organizes. When the transport requests have been organized the TransportCoordinator requests bids from TransportProviders, and will choose the best bid by using a scorefunction. When the best bid have been chosen, a transport task will be created an delegated to the TransportProvider with the best bid. The delegated transport tasks are being monitored in order to replan in case of inconsistency in the plans. The primary objective for this agent is to allocate resources(KVAs) for a given transportaion task and try to have ship blocks collected within the deadline.

- TransportRequestManager: This plugin subscribes to TransportRequest Allocations from the TransportPlanner, and organizes them(with respect to deadline and priority). When a transport request allocation is received this plugin calculates the expected duration of the transportation, places this information in a BidRequest, which is then published.

- ProcessBidRequest: This plugin subscribes to BidRequests an allocates BidRequest Allocations to Transportproviders:
- BidManager: subscribes to BidRequest Allocations in order to monitor the internal state of Bids for consistency, and force replanning when needed. Its also subscribes to TransportProviders in order to chaeck if it has received bids from all TransportProviders before score calculation and publishes the BestBid. The BestBid is calculated with a scorefunction.
- ProcessBestBid: Subscribes to BestBid and publishes an TransportTask Allocation to the TransportProvider with the best bid.
- TransportMonitorPlugin: subscribes to TransportTask Allocations and TransportRequest Allocations, in order to monitor the state of the TransportTask Allocations for internal consistency, checking to see if they are on schedule. The decision regarding replanning of TransportTasks are forwarded to a higher level. In order to perform this action, the TransportMonitor must map TransportTask Allocations to their corresponding TransportRequest Allocations and publish a TransportRequest Allocation Result.

Kamag Vehicle Agent

The KVA is the worker agent, which performs all transportations of ship blocks, and we will divide the behaviour of this agents into the following plugins.

- ProcessBidRequestPlugin: subscribes to BidRequest Allocations and the Environment, checks its plan to see if it is capable of solving the task, and calculates when it can be at the pick up location. This plugin publishes RouteTimeRequests and BidRequest Allocation results.
- BidMonitorPlugin: subscribes to BidRequest Allocations, monitors the results, and checks if they violate the current schedule, publishes updated BidRequest Allocation Results.
- TransportTaskManagerPlugin: subscribes to TransportTask Allocations, and publishes a Transport tasks.
- TransportMonitor: subscribes TransportTask Allocations and checks if they are on schedule. When status on a transportation changes, it publishes a TransportTask Allocation Result.
- EnvironmentLDMPlugin(Gathering): populates the blackboard with the Environment, which is a graph representation of the OSS domain.

- KamagLDMPlugin(Gathering): populates the blackboard with a KamagAsset, representing the KAMAG vehicle with its physical properties, such as speed and maximum payload capacity.
- PerceptionPlugin: Subscribes to the Environment object and RoutePlans. This plugin publishes Perceptions, when ship blocks are detected in the route, or when another KVA is detected.
- VehicleControllerPI: This is the logic controlling a kamag vehicle. This plugin subscribes to the following objects: KamagStatus, RoutePlans, Perceptions and Environments objects. Furthermore this plugin subscribes to TransportTask tasks used for allocating tasks to the KamagAsset(Representing the physical vehicle) and requesting a RoutePlan object. This plugin publishes RouteRequests and SimulationMessages with orders like DRIVE_TO (some location), PICK_UP(a ship block) and PUT_DOWN().
- NegotiationPlugin: This plugin is used to resolve route intersection conflicts between agents. This plugin subsribes the environment and route plans, and allocates the agents route plans to a perceived agent. They will negotiate to find out who has the lowest priority and the shortes path. The agent with the lowest priority will move (publishing a change to the route plan), or the the agent with the shortest path to a free location will move in case of equal priority. The negotiating plugin will publish a new route, and a simulation message, requesting the simulation model to generate an event after a certain duration, whereafter the conflict will be resolved and the KVA can continue on its route.
- SimulationCommunication: This plugin serves to exhange data with the simulation model. When it is created it registeres with a communication broker, that handles the mailbox of this plugin. This plugin subscribes to SimulationMessages.
- RoutePlannerPlugin: Det purpose of this plugin is to generate routes from one location to another location. It subscribes to RouteRequest and RouteTimeRequest, and publishes a RoutePlan or a RouteTimeReport.

12.6 Publish/Subscribe Analysis

The purpose of the publish/subscribe analysis is to ensure that the plugins in the various agents publish and subscribe consistent sets of objects. We conduct this analysis by creating a table as the one shown in table 12.3.

We have now enumerated all plugins required in our OSS society:

Agent	Plugin	Function	Template
Planner	PlannerGUI	Provide UI for the user to create initial TransportRequest tasks	UI-Plugin
	TransportRequest- Allocator	Allocate TransportRequest to TransportCoordinator	Allocator
	AllocationAssessor	Monitors the status of transportation requests for consistency	Assessor
	MonitorGUI	Provide status of every TransportRequest in a GUI	UI-Plugin
Coordinator	TranportRequestManager	Manage TransportRequest from TransportPlanner, publish BidRequest task	Expander
	ProcessBidRequest	Allocate BidRequest task to TransportProvider	Allocator
	BidManager	monitor the state of all Bids, publishing BestBid, when all have replied	Assessor
	ProcessBestBid	Allocate TransportTask to TransportProvider	Allocator
	TransportMonitor	checks transportations for consistency. Publish TransportRequest Allocation Result	Assessor
Kamag Vehicle	ProcessBidRequest	Processes BidRequests from TransportCoordinator	Expander
	BidMonitor	checks bids for consistency	Assessor
	TransportTask- Manager	Create TransportTask tasks	Expander
	TransportMonitor	checks transportations for consistency	Assessor
	EnvironmentLDM	Create Environment	LDM
	KamagLDM	Create KamagAsset	LDM
	Perception	perceives the environment.	Custom
	VehicleController	Allocate TransportTask to local asset	Allocator
	Simulation- Communication	Exchange data with the simulation model	Custom
	RoutePlanner	calculate shortest path between two locations	Custom

Table 12.2: Plugins required in the OSS society

By means of this table, it is clear that we have no overlabs or gabs in the coverage of publishes and subscribes between plugins in any agent.

12.7 Task Grammer

In this step, we detail the content of the tasks produced and consumed by the various plugins.

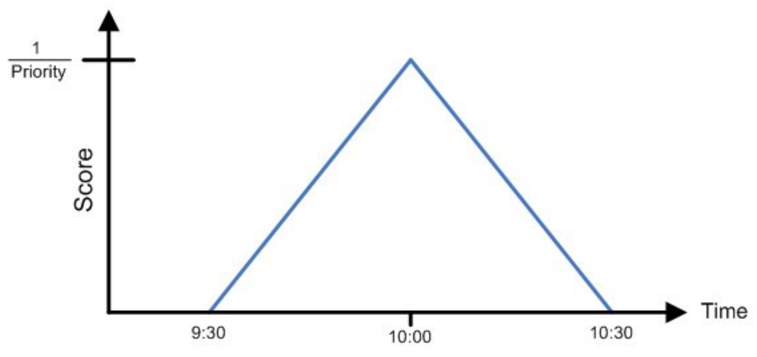


Figure 12.4: Deadline scoring function : "At 10:00"

All tasks has the same type of direct object (content of the task), since it essentially is the same task in different wrapping. Thus the TransportAsset must be able to contain all information necessary to process the task. Note that all preferences have an START_DATE, which contains the start time or deadline of a task. When coordinating the tasks the coordinator uses the score function shown in figure 12.4. The score function makes it possible to reason about bids from KVAs, thus when coordinating tasks, the ones with the highest priority will automatically have an larger score function value, than tasks with lower ones, thus securing that tasks that are most urgent has the highest priority when coordinating tasks.

12.8 Plan Element Map

In this section we lay out map of the different PlanElements in the various agents, to see how incoming tasks are utimately handled. Figure 12.5 reflects how incomming tasks are handled in the PA. The PA publishes this task, for every transportation request in the PlannerGUI, and allocates a plan element to the organizational asset representing the entity with the TransportCoordinator.

Agent	Plugin	Publishes	Subscribes
Planner	PlannerGUI	TransportRequest	
	TransportRequest-Allocator	Allocation (TransportRequest)	TransportRequest
	AllocationAssessor	Report	Allocation (TransportRequest)
	MonitorGUI		Report
Coordinator	TaskRequest- Manager	BidRequest	Allocation (TaskRequest)
	ProcessTransport-Request	Allocation (BidRequest)	BidRequest
	BidManager	BestBid	Allocation (BidRequest)
	ProcessBestBid	Allocation (TransportTask)	BestBid
	TransportMonitor	Allocation Result (TransportRequest)	Allocation (TransportTask)
Kamag Vehicle	ProcessBidRequest	RouteTimeRequest, Allocation Result (BidRequest)	Allocation (BidRequest), Environment object
	BidMonitor	Allocation Result (BidRequest)	Allocation (BidRequest)
	TransportTaskManager	TransportTask tasks, Allocate(TransportTask) to local asset	Allocation (TransportTask)
	TransportMonitor	Allocation Result (TransportTask)	Allocation (TransportTask)
	EnvironmentLDM	Environment object	
	KamagLDM	KamagAsset	
	Perception	Perception object	Environment objects, RoutePlan objects
	VehicleController	SimulationMessage objects	TransportTask tasks, KamagStatus objects, RoutePlan objects, Perception objects
	Simulation-Communication	KamagStatus objects	SimulationMessage objects
	RoutePlanner	RoutePlan objects, RouteTimeReport objects	RouteTimeRequest objects, RouteRequest objects

Table 12.3: Publish/subscribe analysis

Verb	Direct Object	prepositions	Aspects/Preferences
TransportRequest	TransportAsset		START_DATE(10:00)
TaskRequest	TransportAsset		START_DATE(10:00)
BidRequest	TransportAsset		START_DATE(10:00)

Table 12.4: Task Grammar

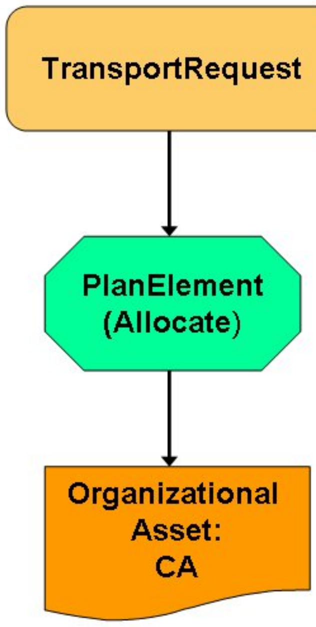


Figure 12.5: Planelement Map for the PA

When the CA receives a BidRequest task or a TransportTask task (published internally in the agent), it allocates the cooresponding plan element the organizational asset representing the entity with TransportProvider role.

When the KVA receives a TransportTask (published from within the agent) then it allocates a planelement to its local KamagAsset, which represents the physical KAMAG vehicle in the system.

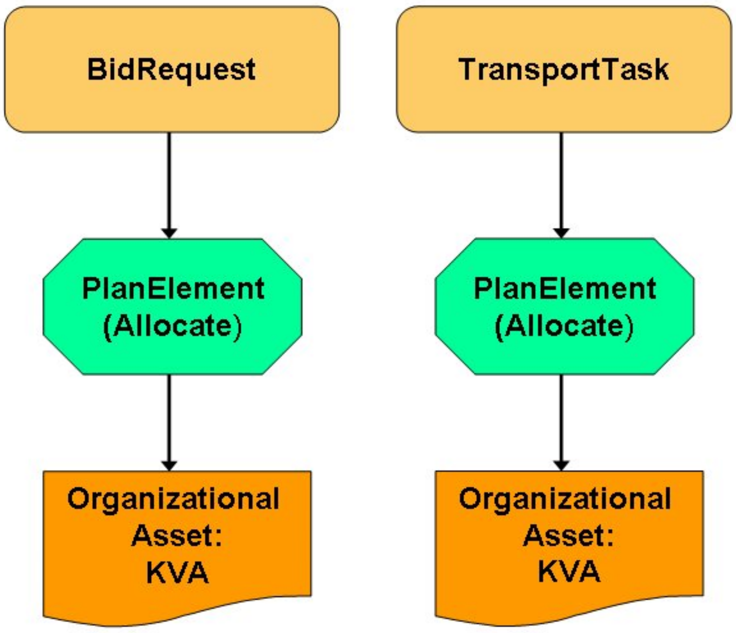


Figure 12.6: Planelement Map for the CA

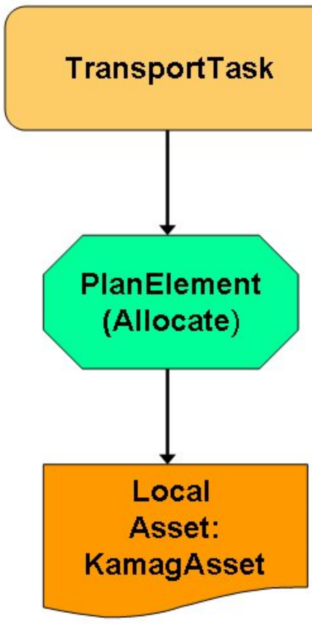


Figure 12.7: Planelement Map for the KVA

12.9 Asset/Propety Analysis

We will now state the assets and properties being used in the agents. We will start by defining the properties:

- VehiclePG: A vehicle has a speed and a maximum speed.
- ContainerPG: A container has a payload and a maximum payload.
- TransportPG: A transport has a block number, a pick location, put down location, a deadline, a priority, a duration, a status and a scorevalue.
- ShipBlockPG: A ship block has a block number(type identification), a weight, a height, length, and a grandblock family.

Now we are ready to define the assets:

- KamagAsset: Represent a KAMAG vehicle, with container and vehicle properties.
- TransportAsset: Represents a transportation request from a C-planner including transport properties
- ControlPointAsset: represent a control point in the simulation model or in the environment, thus it must have storage, supply, equipment, paint and shipblock properties.

12.10 Execution Monitoring/Dynamic Replanning

So far we have discussed planning in a static way. If no requirements change and every transportation task gets allocated the KVAs then there is no need for replanning. The allocations between the agents in the society are shown in figure 12.8. We will now describe the dynamics in the OSS society.

When a C-planner uses his GUI to request transportations he activates his PA. The PA is a reactive agent, which do nothing more than pass transportation requests on to the organizational asset representing the CA. Viewing the agent communication from an external view as the one in figure 12.8, it is seen that the PA can replan, and the CA can replan.

The CA keeps track of the bids from any agent and replans if the bids changes at some point, such that another KVA has a better score function then the one the CA delegated the task to. Thus the task is removed from the other agent.

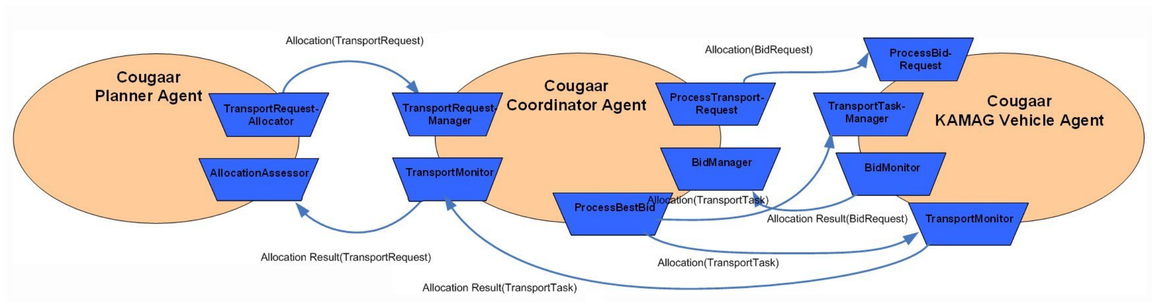


Figure 12.8: Dynamic replanning and execution monitoring

12.11 Node analysis

The node analysis is the last phase of the Cougaar Design Process, and this section deals with the allocation of Agents into nodes. The process of dividing Agents into nodes has no impact on the internal Agent design, since the agents are not aware of in which node they reside, and what agents are co-resident with them in the same node.

Since all agents from the same node, share CPU, memory pool, disk and compete for incoming and outgoing bandwidth traffic, agents which will be in frequent or high bandwidth communication, should be candidates for co-locating in the same node. Agents, which communication resolves in a bandwidth bottleneck

We have a relatively small community, and have thus decided to dispose all agents in the same node. The agents could have been placed in nodes corresponding to the geographical displacement of the human actors at OSS, thus given every C-planner their own node, and placing the D-planner in a centralized placed at OSS.

12.12 The final design

We have designed a multi-agent system with the Cougaar framework based on the Cougaar Methology, in order to solve the logistic problem at OSS. The final design includes a GUI (figure 12.10)for C-planners to order daily transportations, and a GUI for monitoring the results and current status of the transportations. The C-planner have an overview of which transportations are on schedule and which that have failed, meaning deviating to much from the schedule. Based on this information the C-planner at OSS, can replan the transportations that have failed, or change transportations

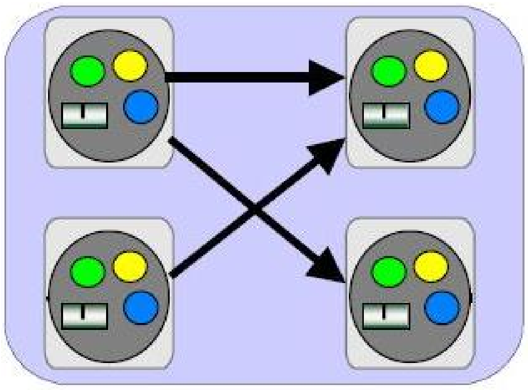
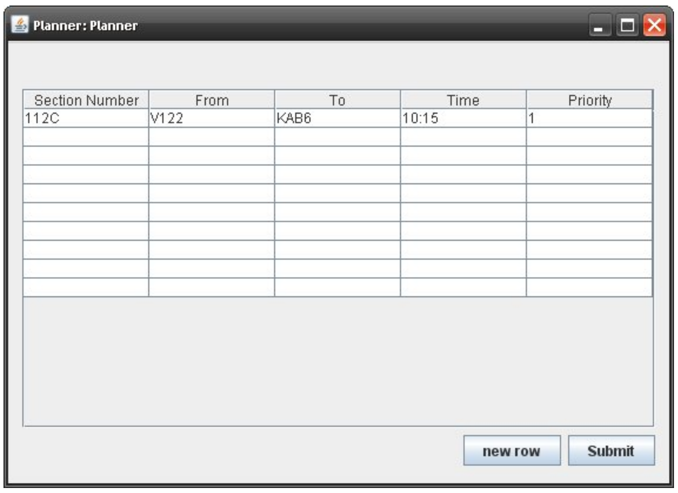


Figure 12.9: Node with Agents

that are still waiting for execution.

KAMAG vehicle agents are now reactive, and react on changes in the environment, which have been implemented with the graph shown in figure 9.4 from section 9.3, and a PerceptionPlugin in the agent, which notifies the agent when ship blocks are perceived on its current route or when other KAMAG vehicle agents are detected in its perception area.



Planner: Planner

Section Number	From	To	Time	Priority
112C	V122	KAB6	10:15	1

Figure 12.10: Planner GUI

Pros	Cons
Very flexible, allowing given a great degree of fredom for programmers	The Cougaar framework is to complex and comprehensive for designing simple multi-agent systems, and for newbie experimentation with the MAS technology
Object oriented: the framework is developed in Java and provides plugins for Eclipse	the framework is asset oriented, limiting the usage of functionality. Some times it is just better to use an object.
It is weel documented: BBN Technologies provides a detailed Cougaar Architecture document, and Cougaar Development document	

Table 12.5: Pros and Cons for the Cougaar framework

12.13 Evaluation of the Cougaar framework

The Cougaar is a very complex and comprehensive MAS frameworks, and it shows that it has been developed over more than 8 years. There are a lot of detailed documentation, such as the “Cougaar Architecture Document”, and the “Cougaar Developers Guide” provided BBN Technologies, and training tutorial at the cougaar community.

information is lacking, e.g. a more developer friendly explanation on how to generate Asserts and property groups (PGs), since they are an important part of the cougaar methology. We have explained how to generate assets in section 14.4.1.

Chapter 13

Middleware

In this chapter we will discuss the design of the middleware. The complete system that we are going built was shown in figure 6.3 in section 6.2. We will use a use-case driven design for the middleware and the starting point for the design of the middleware application will be the user who starts the middleware application. The state diagram for the user starting the middleware application can be seen in figure 13.1.

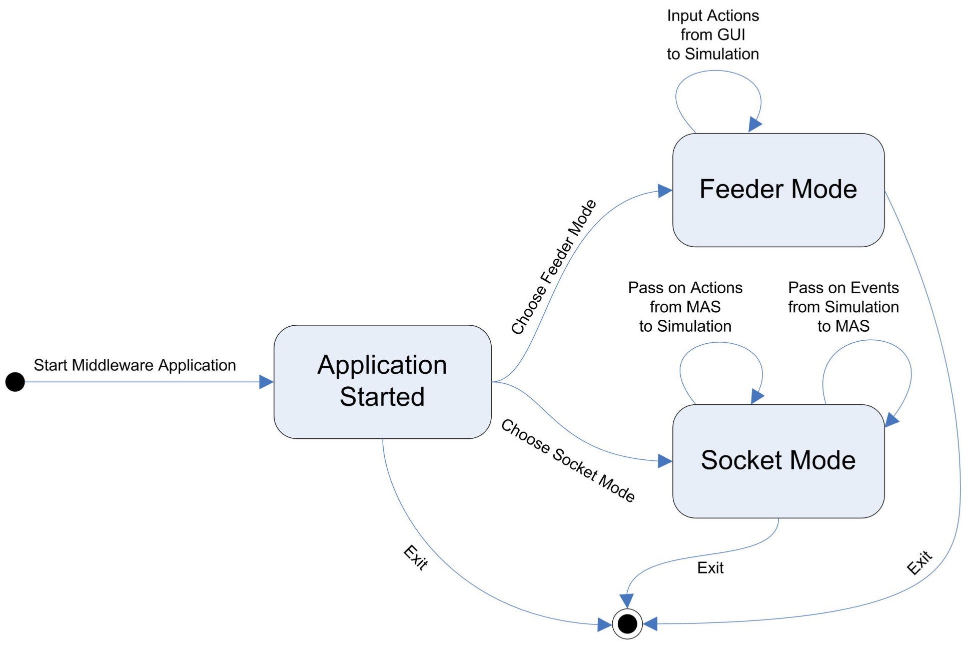

Input Actions from GUI to Simulation
Feeder Mode
Choose Feeder Mode
Pass on Actions from MAS to Simulation
Pass on Events from Simulation to MAS
Start Middleware Application
Application Started
Socket Mode
Choose Socket Mode
Exit
Exit
Exit

Figure 13.1: State diagram for the Middleware Application

When the user has started the middleware he has to choose to one of the following modes to use the middleware:

- **Feeder Mode:** The user inputs actions to the simulation
- **Socket Mode:** The MAS sends actions to the simulation and receives events from the simulation.

The GUI window for choosing the desired mode can be seen in figure 13.2

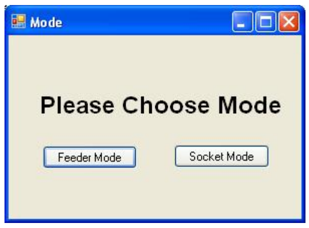


Figure 13.2: Middleware Mode Selection GUI Window

In the next sections these two modes will be described and designed.

13.1 Feeder Mode

The Feeder is used to input different kind of actions, which then can be seen in the simulation model. The feeder is used by an human actor. The use cases for the "User"can be seen in figure 13.3

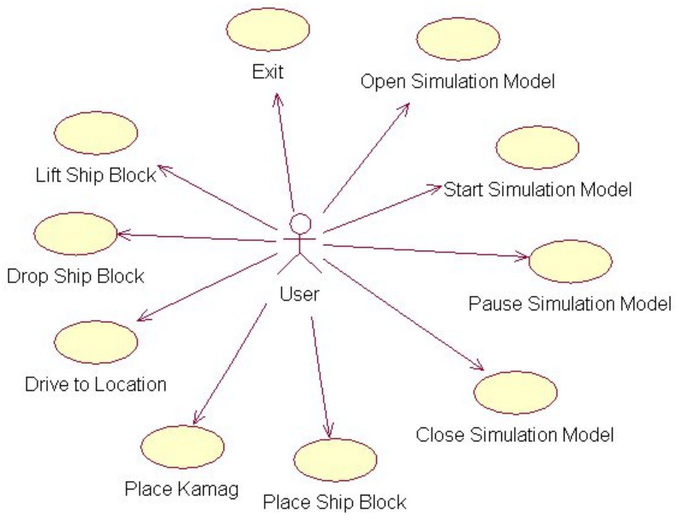


Figure 13.3: Feeder Mode use cases for the User

The Feeder has a GUI where the human actor (User) will be able to execute his use-cases. This GUI can be seen in figure 13.4

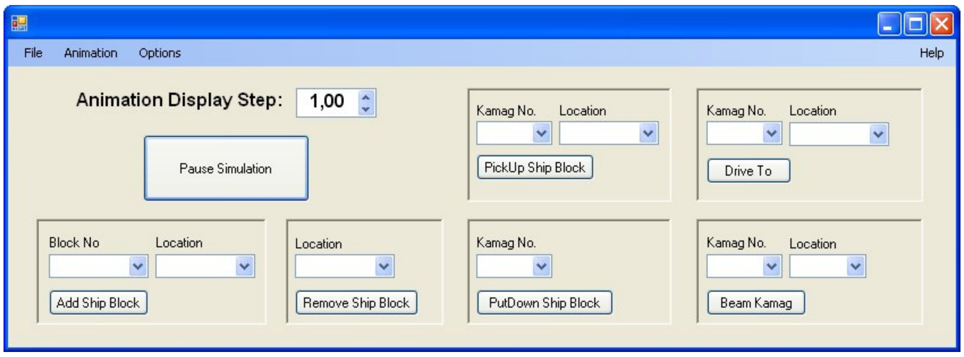


Figure 13.4: Middleware Feeder Mode GUI Window

A short description of each use-case will be described next:

- **Open Simulation Model:** The user opens the simulation model by selecting an Automod builded exe file, and opens it.
- **Start Simulation Model:** The simulation can be started with the "start simulation" button, but only after the simulation model has been opened.
- **Pause Simulation Model:** The simulation can be paused with "pause" button, but only when the simulation has been started. The pause button is the same button as the start simulation button.
- **Close Simulation Model:** The user closes the simulation model by selecting file on the menu-bar and then close simulation. The User can the open a new simulation without restarting the middleware application.
- **Place Ship Block:** The user can place a ship block into the simulation by choosing a ship block no. and a location. The chosen ship block will be placed into the simulation at the specified location, when the user presses the "add ship block" button.
- **Place Kamag:** The user can place a KAMAG from in a location in the simulation by choosing a KAMAG and a location. The chosen KAMAG will be placed when the user presses the "place kamag" button.
- **Drive To Location:** With this use case the user can choose a KAMAG and a location. Then by pressing the "drive to" button, the KAMAG will drive to the chosen location in the simulation.

- **Drop Ship Block:** The user chooses a KAMAG and presses the "drop block" button to drop down the ship block the chosen KAMAG carries at the current location.
- **Lift Ship Block:** The user chooses a KAMAG and presses the "lift block" button to lift up the ship block located at the current location of the KAMAG.
- **Exit:** This use case will exit the Middleware application and close an open simulation model as well.

13.2 Socket Mode

If the user chooses the Socket Mode (see figure 13.1) when the middleware application has started, then the user will see the GUI window shown in figure 13.5

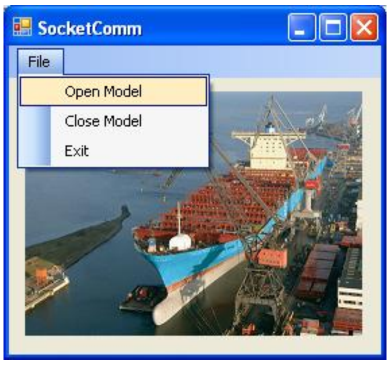


Figure 13.5: Middleware Socket Mode GUI Window

The user can only open and close a simulation model and exit the middleware application with this GUI window. The uses cases for the user in this mode are thus shown in figure 13.6

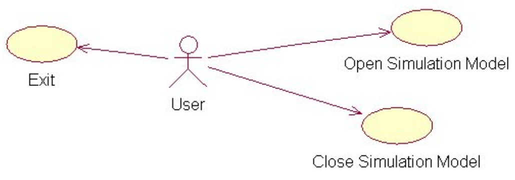


Figure 13.6: Socket Mode use cases

In this mode the Middleware will receive actions from the MAS, and therefore we can see MAS as an actor. In fact there will be an agent in the MAS that will be the actor, acting upon the Middleware. This agent will be called the Communication Agent (see multi-agent design chapter 11). This agent will possess the use cases shown in figure 13.3 minus the use case shown in figure 13.6, that is it will possess the use cases shown in figure 13.7. In fact this agent has a Monitor use case which is used by Communication agent to monitor all kind of activity in the simulation; e.g. when a KAMAG arrives to a location or a KAMAG drops down a ship block and etc. All this kind of events in the simulation environment will be monitored by the Communication agent, which then will inform the corresponding agent in the MAS. To give an example lets look at the case when a KAMAG agent in the MAS drives from one location to another. A state diagram in this case, for the Communication agent, can be seen in figure 13.8

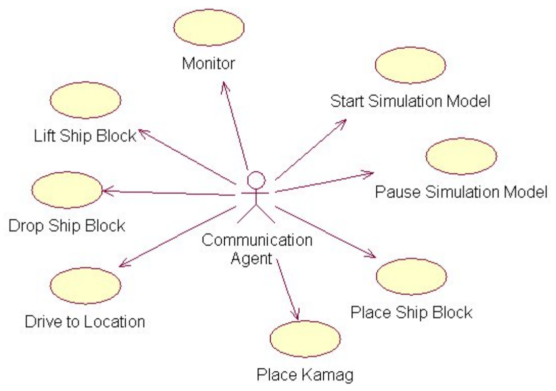


Figure 13.7: Socket Mode use cases

In the first state diagram the Communication agent receives a request “drive to location” from the KAMAG agent. The Communication agent will then send the action “drive to location” to the middleware, which will activate the KAMAG vehicle in the simulation, where the vehicle will begin to move.

The next state diagram shows the state diagram for the use case Monitor. When the Communication agent starts it will enter the monitoring state, and whenever one of the following events occurs in the simulation,

- A KAMAG vehicle arrives to a destination
- Ship block is dropped in a location by a KAMAG
- Ship block is lifted up from a location onto a KAMAG

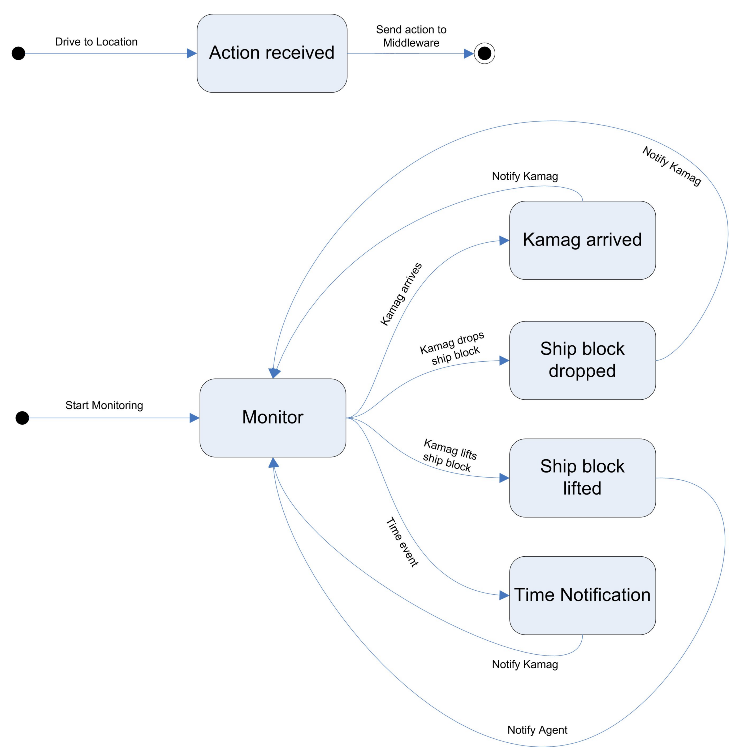


Figure 13.8: Drive to Location State diagram for Communication agent

- A MAS requested time has been reached.

the Communication agent will notify the concerning agents. In our example the Communication agent will then notify the KAMAG agent, when the KAMAG vehicle arrives to the destination in the simulation. The KAMAG agent will then update its state accordingly.

Now lets take a look at the class diagram for the middleware in the next section.

13.3 Class Diagram

The class diagram for the middleware can be seen in figure 13.9.

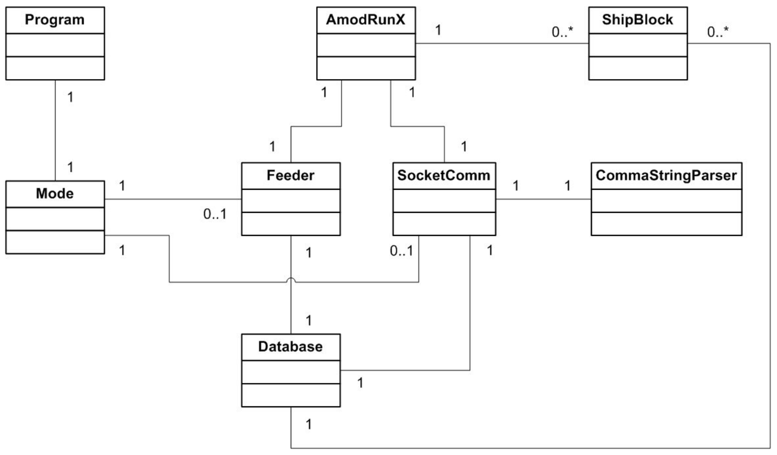


Figure 13.9: Middleware Class Diagram

A short description of each class is given in tabel 13.1.

For further information about each class with its attributes and methods, please look at appendix I

Class	Description
Program	Contains the main method and is the class whith which the Middleware application is started.
Mode	Genrates the GUI window shown in figure 13.2, where the user can select in which mode the middleware application shall be run.
Feeder	Generates the GUI window shown in figure 13.4. The user can then control the simulation and give input to it from the feeder. The input to the feeder are executed in the simulation by the use of the AmodRunX class.
SocketComm	Generates the GUI window shown in figure 13.5. This class receives simulation commands from the Socket Agent in the MAS, parses these commands with the help of the CommaStringParser class and executes these simulation commands by the use of the AmodRunX class. This class also sends received simulation events in the AmodRunX class to the MAS.
AmodRunX	This class contains the Automod ActiveX Object and is used by the Feeder class and SocketComm class to the send simulation commands to the simulation. These simulation commands are for example, to place a ship block at a location, to order a KAMAG vehicle to drive to a location, etc.(see the use cases in figure 13.3). This class also receives generated events in the simulation, e.g. when a KAMAG arrives to a location in the simulation, then this information is sent to the MAS, with the use of the SocketComm class.
Database	Connects to the SQL database, which contains information about ship blocks, KAMAG vehicles, and locations at OSS.
ShipBlock	Whenever information about a ship block is extracted from the database, the information is encapsulated in a ShipBlock object.
CommaStringParser	Is used by SocketComm and AmodRunX classes to parse simulation commands and events. For implementation specific details see section 14.2.3

Table 13.1: Middleware Class Descriptions

Part IV

Implementation and Test

Chapter 14

Implementation

In this chapter we will describe the implementation that are essential to the project. The actual code for the implementation are placed in appendix K and necessary references to this appendix are made.

14.1 AutoMod

The simulation model contains two main types of processes, which are the logic for a queue and the logic for controlling a vehicle.

14.1.1 Queue logic

The code for a storage location is shown below. See figur 10.14 for a flow-diagram.

```
begin
  wait for 10 sec
  move into Q_V122

  if pm.CP_V122 vehicle list size >0 and pm.CP_V122 path distance
      to pm.CP_V122 vehicle list first = 0 then
  begin
    if pickUp = true and kamagPtr = pm.CP_V122 vehicle list first
        then
    begin
      set pickUp to false
      set transportFrom to pm.CP_V122
      wait for 20 sec
      move into pm.CP_V122
      call F_blockPickedUp(kamagPtr, pm.CP_V122);
      send to P_kamagManager
    end
  if exit = false
    send to P_V122
end
```

Furthermore in the painting queues we also have the following line of code included in the second if-block:

```
set this load color to nextof(ltblue,magenta,red)
```

14.1.2 Vehicle logic

The AutoMod code for the process logic in a vehicle is shown below. Se figure 10.13.

```
begin
/*
Dummy load. When the vehicle should drive to a specified
    location.
*/
if load type = L_dummy then
begin
move into kamagPtr current location
travel to transportNext
call F_arrivedToDestination(kamagPtr, transportNext)
end

/*
Ship block. When the vehicle is transporting a block.
And has to drive with the block.
*/
else
begin

if transportNext != null and transportNext != kamagPtr current
    location then
begin
travel to transportNext
call F_arrivedToDestination(kamagPtr, transportNext)
end

/*Check if load should be dropped*/
if transportNext != null and  transportNext = kamagPtr current
    location and putDown = true then
begin
call F_blockDroppedDown(kamagPtr, transportNext)
send to F_getProcessPtrFromLoc(kamagPtr current location)
end

/* Load shall not be dropped, repeat process */
else
begin
wait for 30 sec
send to P_Kamag4843
end
end
end
```

14.1.3 Time event generation

The AutoMod model has a clock, which defines the time in the simulation model. This time needs to be same in the multi-agent system, so that the agents can carry out their planned schedules at the correct times. We have therefore first tried to synchronize the time in the multi-agent system with the simulation model, by sending a socket message with the current simulation time at regular intervals[1]. This didn't succeed very well, because it made the simulation run slower. We therefore chose another approach, the eventbased approach. Whenever an agent makes a schedule, and wants to be notified at a specific time according to that schedule, the function F_generateTimeEvent needs to be called with that time. This function takes two arguments (Hour and Minute), creates a dummy-load and sends it to the process P_generateTimeEvent. The function F_generateTimeEvent is shown below:

```
begin F_generateTimeEvent function

  /* System default time unit is seconds */
  set LPtr_dummy eventTime to ARG_eventHour*60*60 +
      ARG_eventMinute*60 − ac

  clone 1 load of LPtr_dummy to P_generateTimeEvent

return 1
end
```

The process P_generateTimeEvent waits until the specified time has been reached, and notifies the multi-agent system. The code for this process is shown below:

```
  wait for eventTime

  print "timeEvent" "," ac to V_tempString
  call FireUserEvent(0,V_tempString)
```

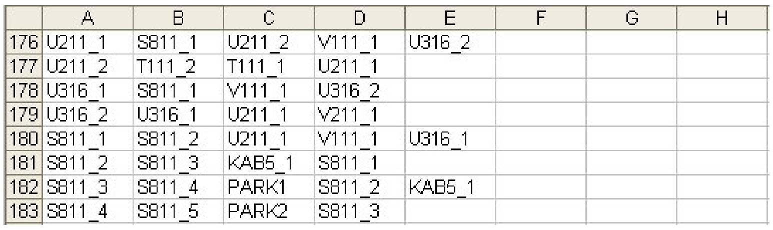
14.2 Middleware

14.2.1 Distance calculation of neighboring control points

The A-star algorithm uses distance costs between a control point and its neigbours. We have therefore by looking at figure 9.4, taken each control point and written its neighbours in a table in an Excel-file[2]. A part of the table are shown in figure 14.1.

[1]We tried and interval of 1 minute, i.e. we sent a time message for every simulation minute

[2]The file called ControlPoints.xls



	A	B	C	D	E	F	G	H
176	U211_1	S811_1	U211_2	V111_1	U316_2			
177	U211_2	T111_2	T111_1	U211_1				
178	U316_1	S811_1	V111_1	U316_2				
179	U316_2	U316_1	U211_1	V211_1				
180	S811_1	S811_2	U211_1	V111_1	U316_1			
181	S811_2	S811_3	KAB5_1	S811_1				
182	S811_3	S811_4	PARK1	S811_2	KAB5_1			
183	S811_4	S811_5	PARK2	S811_3				

Figure 14.1: Part of control point neigbour table

From the figure we can see that control point "U211_1" has the neigbours "S811_1", "U211_2", "V111_1", and "U316_2".

In Excel, the created table, is then saved as an XML-file, so that we can process each control point. A part of the XML-file is shown below.

```
<Row>
  <Cell><Data ss:Type="String">U211_1</Data></Cell>
  <Cell><Data ss:Type="String">S811_1</Data></Cell>
  <Cell><Data ss:Type="String">U211_2</Data></Cell>
  <Cell><Data ss:Type="String">V111_1</Data></Cell>
  <Cell><Data ss:Type="String">U316_2</Data></Cell>
</Row>
<Row>
  <Cell><Data ss:Type="String">U211_2</Data></Cell>
  <Cell><Data ss:Type="String">T111_2</Data></Cell>
  <Cell><Data ss:Type="String">T111_1</Data></Cell>
  <Cell><Data ss:Type="String">U211_1</Data></Cell>
</Row>
<Row>
  <Cell><Data ss:Type="String">U316_1</Data></Cell>
  <Cell><Data ss:Type="String">S811_1</Data></Cell>
  <Cell><Data ss:Type="String">V111_1</Data></Cell>
  <Cell><Data ss:Type="String">U316_2</Data></Cell>
 </Row>
```

When the user selects[3] "Add distances from XML file" in the Feeder as shown in figure 14.2, the code shown in K.1 will be executed. This code will generate a hashmap, where the key is a control point, and the value is a list of control points(the value represents the neighbours of the key). At last the method "database.insertDistances(controlPoints);" will be called, which is shown in section K.2. This method will find the distances between all neighbouring control points and insert them to the MySQL database.

14.2.2 Coordinate Calculation of Control Points

To use the A-star algorithm for finding the shortest path, between two Control Points in the system, we have to know the coordinates of all the con-

[3]Can only be selected when an AutoMod Model has been opened

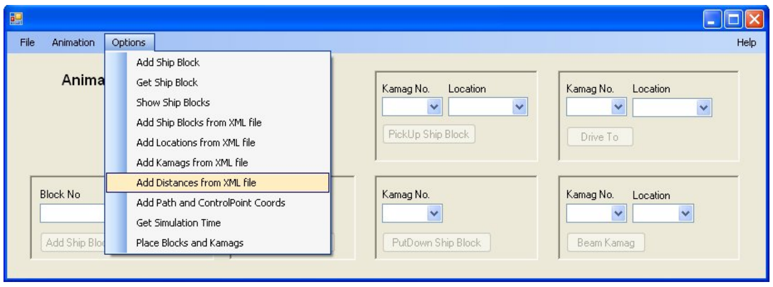


Figure 14.2: GUI window to add neighbours and their distances

trol points. A control point is located on a path in the AutoMod simulation model. An example of a path system with belonging control points is shown in figure 14.3

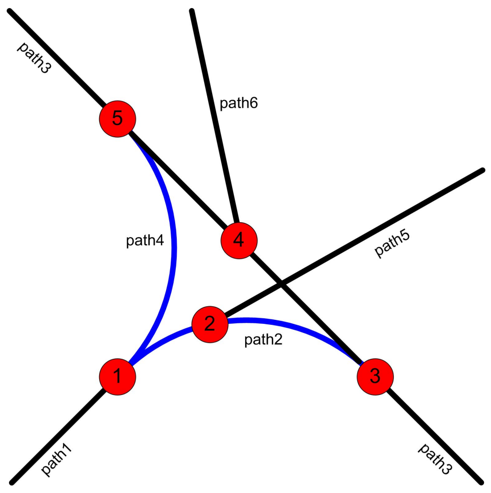


Figure 14.3: Example path system with control points

In this figure we see six path's (the lines) and five control points (the red spots). The two of the path's which are blue, namely path2 and path4 have the form of an arc, the other four path's are straight lines.

The properties of a path is given next:

- **a path** has a unique name
- **a path** can be one or two directional (in our model all the paths are two directional)
- **a path** can have the form of an arc or a straight line
- **a path** having the form of an arc has following properties

  - **(begX,begY)**, defines the the start coordinate of the arc
  - **(cenX,cenY)**, defines the center coordinate of the arc
  - **angle**, defines the angle formed from the start point to the end point from the center of the arc (se figure 14.4)

- **a path** having the form of a straight line has following properties
  - **(begX,begY)**, defines the the start coordinate of the straight line
  - **(endX,endY)**, defines the the end coordinate of the straight line

The properties of an arc path and a straight line path are shown respectively in figure 14.4 and 14.5.

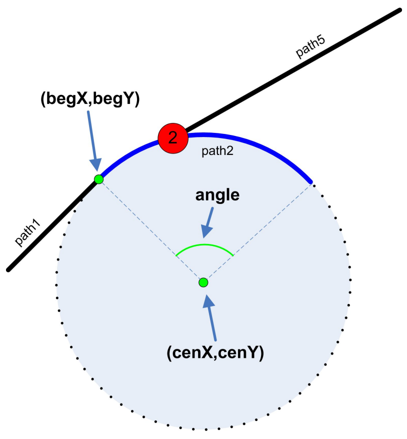


Figure 14.4: Arc path properties

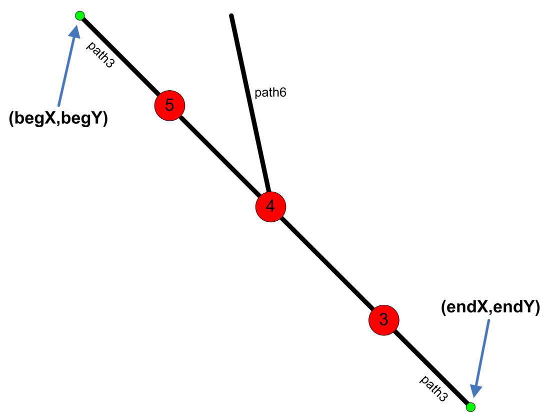


Figure 14.5: Straight line path properties

The properties of a control point is given next:

- **a control point** has a unique name
- **a control point** is located on a path
- **a control point** has a distance, which is measured from the start of the path which it is located on

So the coordinate of the control point in the AutoMod model is not given, only the distance from the beginning of the path is given. We will then calculate the coordinate **(coordX,coordY)** of a control point. The calculation method of the control point coordinate depends on whether the controlpoint is located in an arc path or a straight line path as shown in figures 14.6 and 14.7, where the red spots are control points.

In the next two subsections we will show how to calculate **(coordX,coordY)** when the control point is located on an arc path or on a straight line path

Control point on arc path

Figure 14.6 below shows the different properties of an control point located on an arc path (The path properties are also shown).

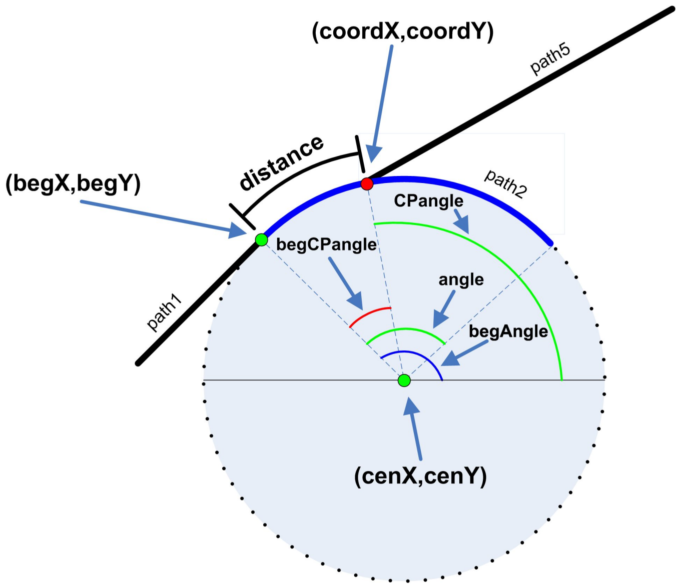


Figure 14.6: Control point located on an arc path

To calculate the coordinate **(coordX,coordY)** of a control point located on an arc path we have to do following:

1. translate **(cenX,cenY)** and **(begX,begY)** so that **(cenX,cenY)** becomes (0,0)
2. calculate the radius and circumference of the circle, which the path is a part of.
3. change the coordinates of **(begX,begY)** to be in the unit circle
4. calculate **begAngle** (see figure 14.6)
5. calculate **begCPangle** (see figure 14.6)
6. calculate **CPangle** (see figure 14.6)
7. calculate the coordinate of the control point in the unit circle **(coordX0Unit,coordY0Unit)** by the help of **CPangle**
8. multiply **(coordX0Unit,coordY0Unit)** with radius to get **(coordX0,coordY0)**

9. translate **(cenX,cenY)** back from (0,0) to original coordinate and translate **(coordX0,coordY0)** accordingly

The code for finding **(coordX,coordY)**, are shown in section K.3

Control point on straight line path

Figure 14.7 below shows the different properties of an control point located on a straight line path (The path properties are also shown).

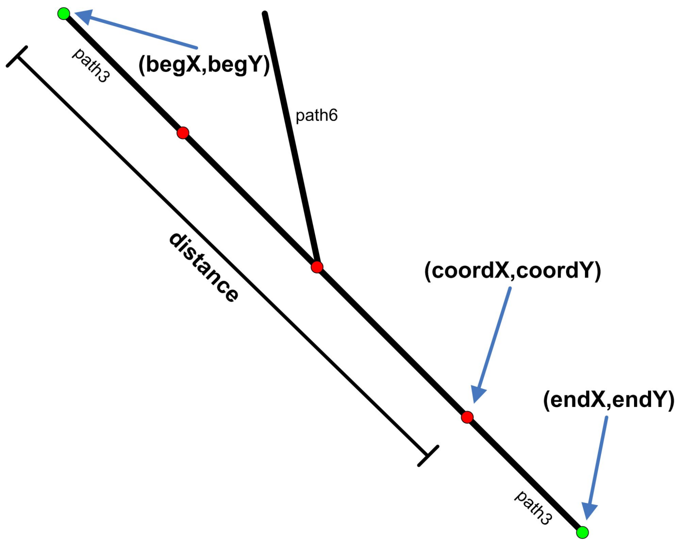


Figure 14.7: Control point located on a straight line path

To calculate the coordinate **(coordX,coordY)** of a control point located on a straight line path we have to use following two equations

$$y = a \cdot x + b \tag{14.1}$$
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{14.2}$$

The first equation is the equation for a straight line, where a is the slope of the line and b is the intersection of the line with the y-axis. The second equation is the equation for a distance between two points in a coordinate system.

The steps in calculating the coordinate **(coordX,coordY)** of a control point located on a straight line path are shown below:

1. If the straight line path is vertical then

- **coordX = begX** and **coordY** is either **begY + distance** or **begY - distance** depending on whether **endY** or **begY** is biggest.

2. Else

   (a) Find a and b in equation 14.1, by using **(begX,begY)** and **(endX,endY)**.

   (b) Insert **(begX,begY)** and **(coordX,coordY)** in equation 14.2 so that we get $d = \sqrt{(coordX - begX)^2 + (coordY - begY)^2}$

   (c) Insert **(coordX,coordY)** in equation 14.1 so that we get $coordY = a \cdot coordX + b$

   (d) Insert $coordY = a \cdot coordX + b$ into $d = \sqrt{(coordX - begX)^2 + (coordY - begY)^2}$ and solve the second degree equation (only **coordX** is unknown). From the two solutions, pick the one that is located inbetween **begX** and **endX**

   (e) Use $coordY = a \cdot coordX + b$, to find **coordY**

The code for finding **(coordX,coordY)**, are shown in section K.4

14.2.3 Middleware communication and reflection

The middleware handles method calls from the Muti-Agent System and the AutoMod ActiveX Runtime object. In the next two subsections, we will look at how the middleware handles a method call from:

1. the Multi-Agent System
2. the AutoMod Runtime Object

Handling the Multi-Agent System

When the middleware is started in "socket mode", it will run a TCP/IP socket server, which will listen for a connection from the Multi-Agent System. The code for this is shown in K.5. This code will establish a connection between the Multi-Agent System and the middleware whenever the Multi-Agent System wishes to make a change in the AutoMod model, by executing a method in the middleware. The Multi-Agent System will send af string containing a method name and some parameters (which will be comma separated) for the method. An example of such a string is shown below.

```
String command = ''driveToLocation, H111, Kamag4843''
```

This string will be parsed as

- **methodName =** "driveToLocation"
- **parameters =** "H111,Kamag4843"

The middleware will then invoke the method driveToLocation with given the parameters. In this case the KAMAG vehicle “Kamag4843” will drive to location “H111” in the AutoMod model.

The methods that can be executed from the Multi-Agent System in the middleware are shown beneath

- **driveToLocation(String destination, String kamag)**
  The specified KAMAG vehicle will drive to the specified location in the AutoMod model
- **pickUpBlock(String location, String kamagNo)**
  The specified KAMAG vehicle will pick up a ship block from the specified location in the AutoMod model
- **putDownBlock(String kamagNo)**
  The specified KAMAG vehicle will put down a ship block into the current location of the KAMAG in the AutoMod model
- **placeBlocksKamags(String blocks, String kamags)**
  The given comma separated strings of ship blocks/locations and KAMAGSs/locations will be used to intialize the AutoMod model, by placing the ship blocks and KAMAGs at thge specified locations.
- **addTimeEvent(String hour, String minute)**
  This method will point out the AutoMod model to generate a timeEvent at the given time

Handling the AutoMod Runtime Object

The running AutoMod model, can also execute methods in the middleware to inform the Multi-Agent System about various changes/information in the model. The methods that the AutoMod model can execute are shown beneath.

- **arrivedToDestination(String kamagAgent, String location)**
  Informs the MAS, that the given KAMAG vehicle (an agent in the MAS), has arrived to the given location.
- **blockPickedUp(string kamagAgent, string location)**
  Informs the MAS, that the given KAMAG vehicle (an agent in the MAS), has picked up a shipblock in the given location.
- **blockDroppedDown(string kamagAgent, string location)**
  Informs the MAS, that the given KAMAG vehicle (an agent in the MAS), has dropped down a shipblock in the given location.
- **timeEvent(String absoluteTime)**
  Informs the MAS about the current time in the AutoMod model.

- **trigger()**
  Triggers the MAS, which will make the D-planner/coordinator agent coordinate the transportation tasks.

In fact AutoMod generates an event (a user event), which contains an integer and a string. This event is captured by the method

```
private void amx\_OnUserEvent(int i, String str)
```

which is shown in K.6. The mentioned methods above are then executed by reflection.

Comma string parser

The Comma string parser is used to extract the method name and the parameters from a comma separated string. The first element in the comma separated string is the method name, the rest are the arguments. The class CommaStringParser has following methods:

- **getArgumentAtPosition:** Returns an argument in the comma string, the position has to bigger than 1 and less than the actual arguments.
- **getArguments:** Returns all the arguments in the comma string.
- **getMethodName:** Returns the method name in the comma string, which is the first element in the comma string.
- **getNumOfArguments:** Returns the number of arguments in the comma string.

The code for the CommaStringParser class is shown in K.7

14.3 DECAF

In this section we will first describe which changes we have made to the DECAF framework and why. After that we will describe the benefits of moving the DECAF framework into Eclipse, which maked it a lot easier to implement the necessary agents, and especially in running the agents.

14.3.1 Modifications to the DECAF framework

The DECAF framework is built in such a way that all the implemented agents and their belonging classes has to be placed at the default package[4], where the DECAF framework with its 55 classes also reside. This structure confuses the programmer, which will have a bad overview of the

[4]the root folder of the source code

classes. We have therefore made a new package called “decaf”, where we have placed the code for the DECAF framework. We have then changed the framework, so that a DECAF agent shall have its own package, instead of beeing placed in the default package.

To fulfill the modifications mentioned, the code below which appears in the class “ExecutorThread.java” at lines 103,168 and 212 have been modified.

```
C = Class.forName(action);
```

The first two occurences of the code above has been replaced with

```
C = Class.forName("decaf"+"."+action); // get the class
```

and the last one has been replaced with

```
String agentDir = "";
if(Local.getAgentName().substring(0,5).equals("Kamag"))
{
  agentDir = "kamag";
}
else if(Local.getAgentName().substring(0,8).equals("CPlanner"))
{
  agentDir = "cPlanner";
}
else
{
  agentDir = Local.getAgentName().substring(0,1).toLowerCase()+
      Local.getAgentName().substring(1);
}
C = Class.forName(agentDir+"."+action); // get the class
```

The code above makes it possible for two or more Kamag (or CPlanner) agents to share the same set of tasks, so that we dont need to make a java package for each of the Kamag (or CPlanner) agents. It can also be seen from the code that the package name for an agent has to start with a lower-case character(e.g. the source code for the DPlanner has to be placed in the dPlanner package, or the framework will not be able to find the tasks/actions that the DPlanner can execute, which will result in a breakdown of the framework).

Planeditor modification

The planeditor needed also some modifications to work correctly after the structure change of the framework. The following code below in the class file “PEFile.java”

```
String JavaCode =
"import java.io.*;\n"+
```

has been rewritten to

```
String packagedir =  dir.getPath();
packagedir = packagedir.replace("src\\", "");
String JavaCode =
"package "+packagedir+";\n\n"+
"import decaf.Agent;\n"+
"import decaf.KQMLmsg;\n"+
"import decaf.LinkedListQ;\n"+
"import decaf.ProvisionCell;\n"+
"import decaf.Util;\n"+
"import java.io.*;\n"+
```

This code is used when the user of the planeditor auto-generates the task-files for the Decaf agents.

14.3.2 Running DECAF from Eclipse with ANT

The Multi-Agent System that we have built with Decaf contains following agents:

1.	CommAgent	6.	Kamag4843
2.	CPlannerHalOst	7.	Kamag4846
3.	CPlannerHalSyd	8.	Kamag4847
4.	DPlanner	9.	Kamag4848
5.	InitAgent	10.	Kamag4849

To run an agent in Decaf, it is required that the ANS is running. The ANS can be started in the command prompt with following command:

```
java ANS
```

Each agent has also to be started in the console, e.g. the "CommAgent" is started as shown below

```
java decaf.Agent -agn CommAgent -p commAgent/commAgent.lsp -gui -go
```

But before this can succeed the user needs to setup the CLASSPATH too. So to simplify all this, we have written an ANT script which can start the ANS and all the agents by simply two mouse clicks (The CLASSPATH is set relative to the eclipse decaf project). The ANT code is can be found in the delivered the CDROM and a screenshot from the Eclipse ANT window is shown in figure 14.8

14.3.3 Message sending in DECAF

DECAF agents execute actions by the use of KQML messages. We have described KQML in section 5.7.1 and will now give a couple of examples from our DECAF implementation.

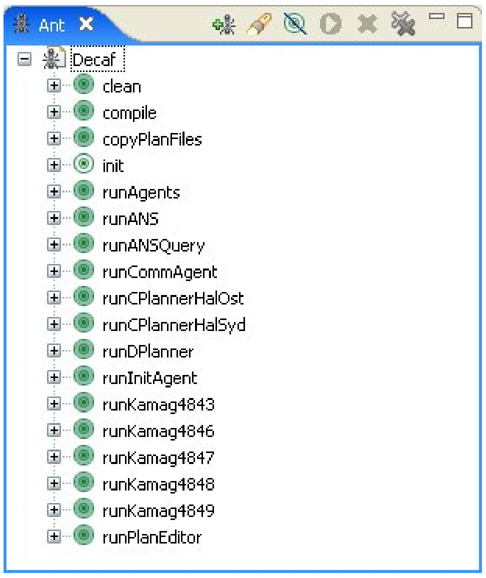


Figure 14.8: Eclipse ANT window

Example 1: The code below is taken from the KAMAG agents's "Kamag_tell" task (se figure 11.2). The "else if" statement is entered, if the KAMAG agent has reached its destination, where it shall drop the ship block it transports. The KAMAG agent will then set its internal state and send a KQML message to the CommAgent, which will inform the AutoMod model, that the KAMAG vehicle shall drop the ship block at the current location.

```
else if(arrivedLocation.equals(toLocation))
{
  Local.userHash2.put("pickUp", false);
  Local.userHash2.put("putDown", true);

  KQMLmsg K = new KQMLmsg();
  K.addFieldValuePair("performative", "achieve");
  K.addFieldValuePair("sender", Local.getName());
  K.addFieldValuePair("receiver", "CommAgent");
  K.addFieldValuePair("language", "DECAF");
  K.addFieldValuePair("ontology", "CommAgent");
  K.addFieldValuePair("content", ":task PutDownBlock "+
  ":kamagNo "+kamagNo);

  return new ProvisionCell(K.getKQMLString(),"OK");
}
```

Example 2: The code below is taken from the DPlanner agents's "DPlanner_DistributeTransports" task (se figure 11.5). The method(action) below is executed whenever the DPlanner agent makes a call for proposal to all the KAMAG agents, to hand over a transportation request from a CPlanner. In the matter of fact, the KQML message below will execute another task "DistributeTransport" located in the DPlanner agents, which then will send the KQML message to each of the KAMAG agents.

```
private void sendCFPToKamags()
{
  String[] kamags = {"Kamag4843","Kamag4846","Kamag4847","
      Kamag4848","Kamag4849"};
  for(int i=0; i<kamags.length; i++)
  {
    KQMLmsg kqml = new KQMLmsg();
    kqml.addFieldValuePair("performative", "achieve");
    kqml.addFieldValuePair("sender", local.getName());
    kqml.addFieldValuePair("receiver", local.getName());
    kqml.addFieldValuePair("ontology", "DPlanner");
    kqml.addFieldValuePair("language", "DECAF");
    kqml.addFieldValuePair("content", ":task DistributeTransport
         :kamagAgent "+kamags[i]+
    ":convID "+convID );
    Util.send(kqml.getKQMLString());
  }
}
```

14.4 Cougaar

We first describe how to generate property groups and assets, then how basic features of the agents are implemented, concerning publishing and subscribing objects in plugins. Futhermore we describe how agents are organized, meaning their awareness of each other in the society, and describe how the customer/provider relationship between agents have been implemented. Moreover we describe how agent-to-agent communication have been implemented, and finalize this section describing the implementation of the environment(the graph from section 9.4) and the A* algoritm for path-finding in the graph.

14.4.1 Property groups and Assets

In this section we describe how Property groups and assets have been implemented in the cougaar framework.

Assets are defined in a asset definition file(def file), e.g. "assets.def". In this file every asset are specified with their name, what property groups they contain, and a describtion (doc). An asset can be composed of multiple property groups, which are defined as slots, e.g. the KamagAsset is composed of three properties groups; VehiclePG, ContainerPG and ShipBlockPG as seen below:

```
[KamagAsset org.cougaar.planning.ldm.asset.Asset]
slots = VehiclePG, ContainerPG, ShipBlockPG
doc=Representation of a Kamag vehicle including representation of
   a vehicle and a container

[TransportAsset org.cougaar.planning.ldm.asset.Asset]
slots = TransportPG
doc=Representation of a transportation reservation

[ControlPointAsset org.cougaar.planning.ldm.asset.Asset]
slots = StoragePG, SupplyPG, EquipmentPG, PaintPG, ShipBlockPG
doc=Representation of a transportation reservation
```

Property groups are specified in properties definition files(def), e.g. "properties.def". In this file all property groups are defined, where the name is specified, e.g. "[VehiclePG]", and the attributes are specified with their name and type, such as "double speedKMH", which indicates that speedKMH is of type double. There is no limitation regarding the number of attributes in a property group. One can specify docs in order to make the generated classes more readable later on. Below are listed the property groups we have implemented in our Cougaar multi-agent system. As seen below, property groups do not need to have any atributes, if their property is describing enough.

```
[VehiclePG]
doc=Contains information regarding the vehicle velocity
slots=double speedKMH, double maxSpeedKMH
speedKMH.doc=The velocity in Km per hour of the vehicle
maxSpeedKMH.doc=The maximum velocity of the vehicle in Km per hour

[ContainerPG]
slots=double payload, double maxPayload
payload.doc=The payload placed in the container
maxPayload.doc=The max payload that can be placed in this
    container

[TransportPG]
doc=Contains information regarding transportation request from
slots= String blockNo, String pickUpLocation, String
    dropOfLocation, Date time, int priority
blockNo.doc = the type identification of the ship block
pickUpLocation.doc = The location to pick up the ship block
dropOfLocation.doc = The location to drop of the ship block
time.doc = The time, at which the ship block should be picked up
priority.doc = The priority of the transport, where a 1 is most
    urgent

[ShipBlockPG]
doc=contains information regarding a shipblock
slots= String blockNo, double blockWeight, double blockLength,
    double blockBreadth, double blockHeight, String grandBlockNo,
    String BlockFamily

[StoragePG] [SupplyPG] [EquipmentPG] [PaintPG]
```

Once we have specified the definition files “properties.def” and “assets.def” we run the PGWriter(with “properties.def” as input) provided with Cougaar to generate the property group java files. After the property group classes have been generated we run AssetWriter(with “assets.def” as input), which generates the assets automatically. Get and set methods for all attributes are automatically generated and the classes are serializeable. Thats all there is to it, now we have generated our property groups and our assets.

14.4.2 Organizing agents

The agents in a Cougaar society are defined in a society file, wherein the relations between the agents are defined. Agents must have an organizational asset representing it in the society. An organizational asset contains information regarding the agents messageaddress, its itemidentification(its name) and a role, which indicates the group the agent belongs to.

We have used customer/provider relationships between the groups in our society, where the customer allocates tasks to providers. Organizational assets are created with the AssetDataParamPlugin class, as shown below. The agent name, the agent messageadress and the itemidentification have been given the same name for simplicity (highlighted in red). In the example below are seen a PA, which has the role "TransportPlanner"(highlighted in blue). The agent below is a customer, thus we do not specify a relationship, since they are specified at providers.

```
<component class="org.cougaar.planning.plugin.asset.
    AssetDataParamPlugin">
<argument>Prototype:Entity</argument>
<argument>ClusterPG:MessageAddress:MessageAddress:Planner</
    argument>
<argument>ItemIdentificationPG:ItemIdentification:String:Planner</
    argument>
<argument>TypeIdentificationPG:TypeIdentification:String:UTC/RTOrg
    </argument>
<argument>EntityPG:Roles:Collection&lt;Role&gt;:TransportPlanner</
    argument>
</component>
```

The relationship is specified at the provider, and the CA has the following extra line in its organizational asset:

```
<argument>Relationship:MessageAddress=Planner,ItemIdentification=
    Planner,TypeIdentification=UTC/RTOrg,Role=TransportCoordinator
    ,StartTime=01/01/2007 12:00 am,EndTime=</argument>
```

As seen above in the relationship we specify the messageadress of the customer(highlighted in red), and the providers role(highlighted in blue). Then the planner can subscribe to organizational assets, that has the "TransportProvider" role, and publish allocation which will then be copied to the providers blackboard. The time indicates when the relationship starts and when it ends, and since the relationship is permanent we dont specify and end time.

14.4.3 Interaction

Our agents have predefined customer/provider relationships and organizational assets representing them. Interaction between agents are implemented with the customer/provider relationship, where customers publishes allocations to their own blackboard, which is then copied to the providers blackboard. An Allocation is composed of a task, a plan and an organizational asset(the receiver/subscribing provider). When a customer publishes an allocation to the blackboard it is automatically copied to the providers blackboard.

```
private void allocateTo(Asset asset, Task task) {
  AllocationResult estAR = null;
  Allocation allocation = ((PlanningFactory)getDomainService().
      getFactory("planning")).createAllocation(task.getPlan(),
      task, asset, estAR, Role.ASSIGNED);
  getBlackboardService().publishAdd(allocation);
}
```

14.4.4 Environment and RoutePlanner

We have used the A-star algorithm for finding the shortest path between two locations in the OSS AutoMod model. This path is used by the KAMAG vehicles to drive from one location to another. The implementation of the A-star algorithm is described in [12], which we have modified to suit our case. The classes and their descriptions are given below:

- **AStarNode** is the abstract class which represents a node in the graph (see the OSS graph in 9.3).
- **AStarSearch** contains the method findPath which takes two nodes as arguments (start and end node), and returns a list of nodes which represents the path, not including the start node.
- **ControlPoint** is the implementation of the AStartNode class. A controlpoint can contain a shipblockPG and/or a KamagAsset. It implements all the abstract methods from the AStartNode class, and has a few get/set methods for its following attributes:
  - *controlPointName:* The name of the node in the graph.
  - *pathName:* The name of the path where this node is located on.
  - *coordX and coordY:* the coordinate of this node in the AutoMod model.
  - *blocked:* A boolean which indicates, whether this node is occupied with a KAMAG vehicle or a ship block.
- **Environment** is the class which contains the OSS graph (the graph in figure 9.4). The class is a singleton, and reads the graph information from a MySQL database into the four attributes of the class when initiated. These four attributes and their responsibilities are as follows:
  - *controlPoints* is a hashtable containing all the controlpoints of the OSS AutoMod model. It maps controlpoint names to AStarNodes, which in fact are ControlPoints.
  - *neighbors* is a hashtable mapping each controlpoint name in the OSS graph to its neighbour controlpoint names

- *distances* is a hashtable mapping each controlpoint in the OSS graph to a hashmap which contains the neighbour controlpoints, which in turn maps a controlpointname to a distance. With this hashmap, one can easy get the distance between two controlpoints, shown by an example beneath

  ```
  (distances.get(''H111'')).get(''V122'');
  ```

- *realDistances* is a hashtable mapping each controlpoint in the OSS graph to a hashmap which contains the neighbour controlpoints, which in turn maps a controlpointname to a distance. The difference between this hashtable and the one above, is that the distances in the distances hashtable are modifed whenever an entity is placed on a controlpoint and when the entity is removed again, then the original distance value is replaced from realDistances hashtable.

This class contains also the important methods, described below:

- public ControlPoint getControlPoint(String controlPointName)
  This method returns a ControlPoint given a controlpoint name as an argument.
- public double getCPDistance(ControlPoint CP1, ControlPoint CP2)
  This method returns the distance between two controlpoints in the OSS graph.
- public List<AStarNode> getCPNeighbors(String controlPointName)
  This method returns a list of neighbour controlpoints given a controlpoint name as an argument.
- public void addObstacleToCP(ControlPoint controlPoint)
  This method increments the distances from the given controlpoint to its neighbours by a virtualdistance=100000, to indicate to the "path algorithm" that this controlpoint is blocked.
- public void removeObstacleFromCP(ControlPoint controlPoint)
  This method decrements the distances from the given controlpoint to its neighbours by a virtualdistance=100000 if the neighbour is not blocked,. This will indicate the "path algorithm"that this controlpoint is now open for passage again.

• **MoreMath** contains different math methods and the one that is used by the A-star algorithm is the sign method. This method returns -1 if the given argument is negative, 1 if it is positive and 0 else.

Chapter 15

Test

Test cases have been performed in order to verify subparts of the overall system. The test cases described in this chapter are all associated with a video that are located in delivered CDROM.

15.1 Simulation

In this section we will set up test cases, which purpose is to test different parts of the simulation model. We will test if we have fulfilled the requirements to the simulation model.

15.1.1 Test case : Create ship block

In this test we will simply test the functionality of creating a ship block and placing it in the simulation model. With this test, we test the ability to create a ship block in the simulation model, and the ability to place it in a given queue, that is visualizing it to the user. This test is performed on the two different ship blocks in table 15.1.

GRANDBLOCK DATA				
GB no.	**Weight [T]**	**Length [m]**	**Breadth [m]**	**Height [m]**
112C	703	17.5	21.1	18.2
214P	262	17,5	27,3	8,2

Table 15.1: Test case ship block

The grandblocks listed in table 15.1 already exist in the OSS database [1] and are therefore accessable from the Feeder.
The test video "testvideo1_1.wmv" shows steel section 112C and 214P from table 15.1 being placed at V122 and V311 respectively. In the video "testvideo1_1.wmv"

[1] see section F.1 for information about this database

it is seen that the blocks and locations are accessable from the Feeder and that the blocks are inserted in the simulation model correctly. When blocks are placed in the simulation model, they stay where they are placed until they are pickup by a KAMAG vehicle or removed.

15.1.2 Test case : Remove ship block

In this test, we test the ability to remove ship blocks from the simulation model. First we create and place ship block 112C and 137C at location V311 and V250 respectively. Then we remove them with the Feeder again as shown in "testvideo1_2.wmv".

15.1.3 Test case : Transport ship block

In this test case we test how transportation is performed. A transportation is divided into the subtasks "pick up ship block", "put down ship block" and "drive to location", which are described in the following subsections. Since theese tasks together form the transportation task, we will combine these tests in one test video.

Test case : drive to a location

In this test, we test a KAMAG vehicle's functionality of driving to a specified destination, such as a storage location or a control point. In the Feeder we choose KAMAG 4843 and the destination B4 and tell the KAMAG to drive without any ship block[2] as seen in test video "testvideo1_3.wmv". Later we tell KAMAG 4846 to drive with a ship block to a location, which works well as seen in the video, but a collision occurs between KAMAG 4843 and 4846. Collission control is not implemented in the simulation model, because this is the responsibility of the system logic, i.e. the multi-agent system.
The test video "testvideo1_3.wmv" has showed that we are fully able to control the KAMAG vehicles in the simulation model externally using the Feeder, and therefore controlling the simulation model from the MAS through the middleware will also work.

Test case : Pick up a ship block with a vehicle

In this test we instruct a KAMAG vehicle to pick up a specified ship block. In the Feeder GUI we select the KAMAG which shall pick up the ship block, and then we specify the location, where the KAMAG vehicle is supposed to pick up the ship block. When the KAMAG vehicle arrives to the storage location, wherein the ship block is stored, the ship block will be picked up

[2]The transparent load that can be seen on the KAMAG vehicle is a "dummy load"

onto the KAMAG vehicle. As seen in the test video "testvideo1_3.wmv" we place three ship blocks at different locations (V311, V315 and V320), and tell KAMAG 4846 to drive to location V320, pickup the ship block and drive to location H111. The KAMAG vehicle correctly pick's up the load from the storage location, where the color of the storage location shifts color from green to red, indicating that the storage location is now empty.

Test case : Drop down a ship block from a vehicle

Here we test wether the functionality of placing a ship block in a storage location from a KAMAG vehicle functions as expected. In order to make a KAMAG vehicle place the ship it carries at its current location, we select the KAMAG vehicle in the Feeder and press the "PutDown Ship Block" button. In "testvideo1_3.wmv" it is seen that KAMAG 4846 places the ship block it carries into location H111 and drives to location H115 without any ship block. So dropping down a ship block from a vehicle works well.

15.1.4 Test case : Painting of ship blocks at painting halls.

In this case we test whether ship blocks dropped at paintings halls get painted. A ship block transported to a painting hall, without beeing dropped, should not be painted. In the video "testvideo1_4.wmv", it is shown that the KAMAG 4843 vehicle transports shipblock 112C to KAB4 and back to location V122 without dropping it. This does not change the painting of the ship block. Then the same procedure is repeated, but this time the ship block is also dropped down and picked up at KAB4. This process has painted the ship block as desired.

15.2 DECAF

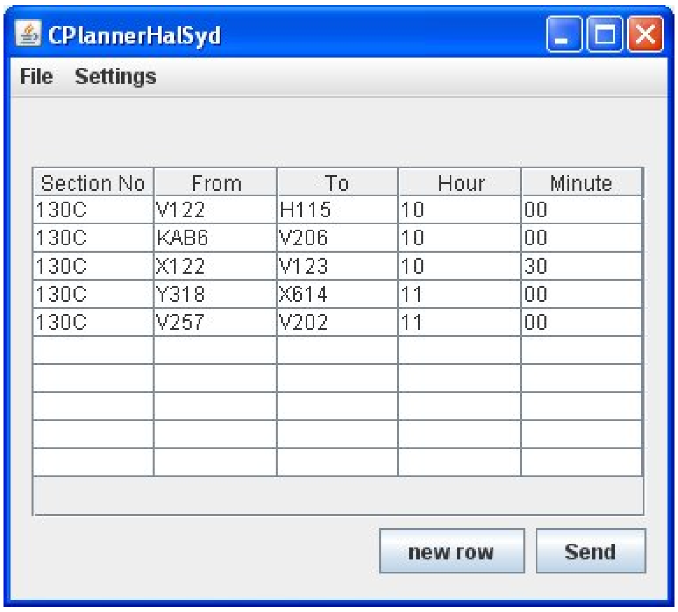
In this section we describe the two cases that we have tested the DECAF implementation with.

15.2.1 Start and initialization

We have used eclipse to implement the MAS, with the use of the DECAF framework. To start the MAS, we have written and ANT script, and in eclipse we start the MAS by first clicking on "runANS" and afterwards on "runAgents". This will start 13 JAVA GUI's, as seen in the movie "testmovie2_1.wmv". We then use the InitAgent to initialize the simulation model, with initial setup of ship blocks and KAMAG vehicles. According to "testmovie2_1.wmv" both the start of the MAS and initialization, succeeds very well.

15.2.2 Requesting transports

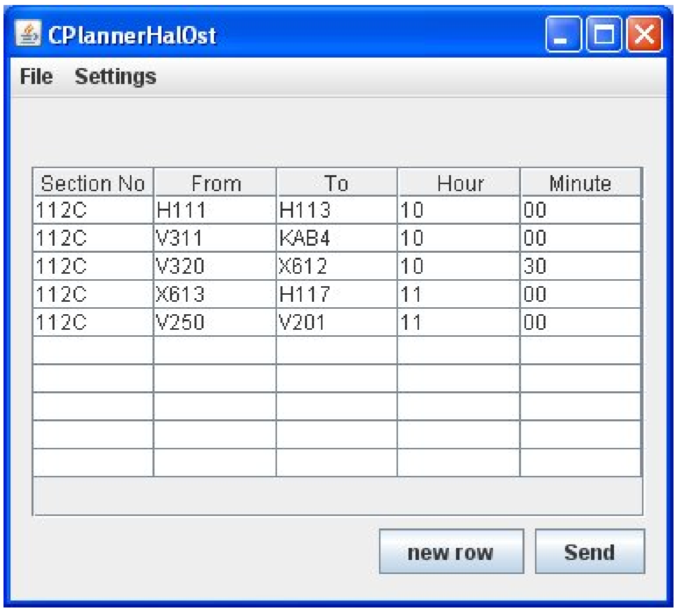
In this test case, the MAS is tested to see whether it is capable of any planning. The test case contains two C-Planner agents, each making five transportation requests. The C-Planner agents and their request are shown in figure 15.1 and 15.2



Section No	From	To	Hour	Minute
130C	V122	H115	10	00
130C	KAB6	V206	10	00
130C	X122	V123	10	30
130C	Y318	X614	11	00
130C	V257	V202	11	00

Figure 15.1: HalSyd C-Planner transportation requests

The planning in our DECAF MAS is implemented in such a way that when C-planner agents requests for transportations, then the D-planner agent collects these tasks in a list. The transportation tasks in this list are sorted by their pick up times, and then by the length of their transportation times. The D-planner is triggered for every ten minutes in the current implementation, which makes the D-planner take the first transportation task from the list mentioned, an tries to delegate this task to a KAMAG agent as described in section 11.2.2.
There is ten transportation tasks to be solved in this case, and only five KAMAG vehicles. The D-planner delegates the first five tasks. It seen in the video that the KAMAG vehicles arrive approximately at the correct pickup times. It can also be seen in the testvideo "testvideo2_2.wmv" that collisions between KAMAG vehicles happen. This is due to the fact that the routing of the KAMAG vehicles aren't designed and therefore not implemented in the MAS. The routing is done by the simulation model, which choses the shorstest path to the pick up location of the ship block, without taking care of other vehicles in its way.



Section No	From	To	Hour	Minute
112C	H111	H113	10	00
112C	V311	KAB4	10	00
112C	V320	X612	10	30
112C	X613	H117	11	00
112C	V250	V201	11	00

Figure 15.2: HalOst C-Planner transportation requests

We can conclude from the video “tesvideo2_2.wmv” that the agents works well[3], and that the implemented MAS solves the logistic case of transporting ten ship blocks. The D-planner coordinates the tasks very well, and the KAMAG vehicles transports the ship blocks at the correct times.

15.3 Cougaar

15.3.1 Test case : controlling basic functionality

In this case we test the basic functionality in the Cougaar framework consisting of three agents, a PA a CA and a KVA, we will instruct the KVA to transport a ship block from one location to another. Thereby also verifying that the middleware can be used to exchange data between the simulation model and the Cougaar society. The test video is called “testvideo3_1.wmv”. This test is only conducted with one KVA, thus only verifying the basic functionality.
We can conlude that the middleware can exchange data between the simulation model and the cougaar society, and that the PA request a transportation from the CA, which forwards the request to the KVA, and that the KVA accepts the task. The KVA picks up the block at its pick-up location, and places the block at its put-down location.

[3]Besides collision aviodance

Part V

Reflection and Future Work

Chapter 16

Discussion and Conclusion

In this chapter we discuss the results, issues and experiences that we have obtained in this thesis. We begin by discussing alternative approaches. Then we present the results from developing the simulation model in AutoMod and discusses issues and experiences with AutoMod. This is followed by presenting the results obtained from the development of the multi-agent system in Decaf and Cougaar respectively, and further evaluates the two MAS frameworks. Then we will present the result of developing the Middleware application, and in the following section we will discuss future work in respect to this thesis project and OSS. We briefly mention the various technologies we have worked on in this thesis. In the last section of this chapter we conclude on the results archieved in this thesis.

16.1 Alternative approaches

We will in this section present alternative approaches, starting with overall approaches, followed by a disscussion regarding alternative organization of the agents.

16.1.1 Overall alternatives

In our thesis project we started by designing a detailed scaled version of the OSS domain in a three dimensional graphical environment, modeling every building, storage facility, as well as the dynamic entities, such as KAMAG vehicles and the ship blocks, in order to simulate and visualize the transportations at the shipyard. After designing the simulation model, we developed a middleware application that included a feeder, which was used to verify the simulation model. Finally we designed a multi-agent system, and extended the middleware, such that data between the simulation model and the multi-agent system could be exchanged.

An alternative approach could be to design a multi-agent system first and then to develope a middleware application without a simulation model. Instead the middleware could contain a virtual simulator as shown in figure 16.1, that could simply contain a lookup table. For example, when the multi-agent system tells a KAMAG vehicle to drive from one location to another, the virtual simulator could check the table, calculate the travel time and generate an event to the multi-agent system, when the travel time has passed.

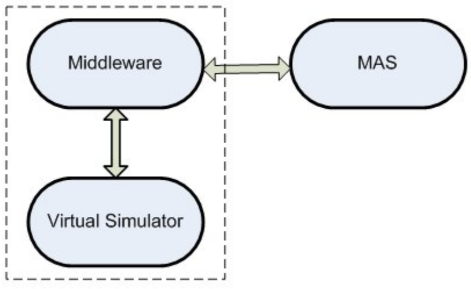


Figure 16.1: Alternative solution: MAS with virtual simulator

A second approach could have been to design the overall system with a complete MAS and a simple simulator either connected directly or through some middleware as seen in figure 16.2. The simple simulation model could simulate a driving vehicle, which given a distance, simulates to drive until the distance has been travelled. The simulator could then generate an event to the multi-agent system. The simple version of the simulator could for instance be designed in a simulation tool like Renque.

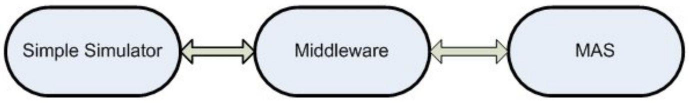


Figure 16.2: Alternative solution: MAS connected with simple simulator

A third approach could have been to use a MAS framework, which has an integrated simulator like Madkit for instance. See figure 16.2.

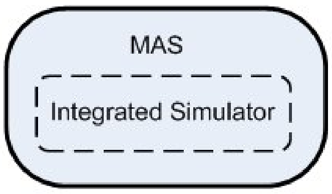


Figure 16.3: Alternative solution: MAS with integrated simulator

16.1.2 Alternative Agent Organization

In this thesis we have organized the agents, such that the human organization at OSS have been preserved, i.e. multiple planner agents[1], a centralized coordinator agent[2] and several KAMAG vehicle agents[3], that represents C-planners, D-planners and KAMAG drivers respectively.

An alternative could be to remove the centralized coordinator agent and thereby letting planner agents coordinate and negotiate directly with the KAMAG vehicle agents. This solution would make the system decentralized, and eliminate a possible centralized coordination bottleneck. The downside with this approach is that the amount of information that would be exchanged would increase exponentially, thus instead of a centralized coordinator negotiating with five KAMAG agents, every planner agent would have to negotiate with every KAMAG vehicle agent.

Another alternative would be to view the transportation problem at OSS from a higher degree of abstraction, e.g. introducing the B-planners[4] as agents in the system. With this approach we would be able to follow every ship block through the entire system, thus introducing the possibility of representing ship blocks as agents in the system, thereby making the ship block agent the central agent in the system. The agent would then have a life cycle starting at its creation (production or arrival) and ending at its final destination at the dock. The ship block agent could then have knowledge regarding its destined flow in the system, and it would be possible for the ship block agent to request different services from other agents, such as the transportation service, we have modeled in this thesis, and further to use supply, equipment and painting agents[5].

[1]C-planner agent in Decaf and Planner Agent in Cougaar
[2]D-planner agent in Decaf and Coordinator agents in Cougaar
[3]KAMAG agents in Decaf and KAMAG vehicle agents in Cougaar
[4]See section 2.2.2 for a description of the B-planner.
[5]Adapted from section 9.1.4

16.2 AutoMod

We have created a 3D simulation model(see figure 16.4) of the Odense Steel Shipyard(OSS) using AutoMod in order to simulate reality. The graphical simulation model is based on a dxf file provided by OSS, which contains information regarding placement of all buidings, the road system, and every storage location. The OSS road system have been designed with an AutoMod Path Movement System, wherein AutoMod control points have been placed according to a graph 9.4. The storage location have been modelled with AutoMod queues, and the graphics have been made with ACE. Each building have been designed independently in ACE. Likewise the KAMAG vehicles in OSS have been modelled as AutoMod Vehicles in the path movement system, with graphics designed in ACE.

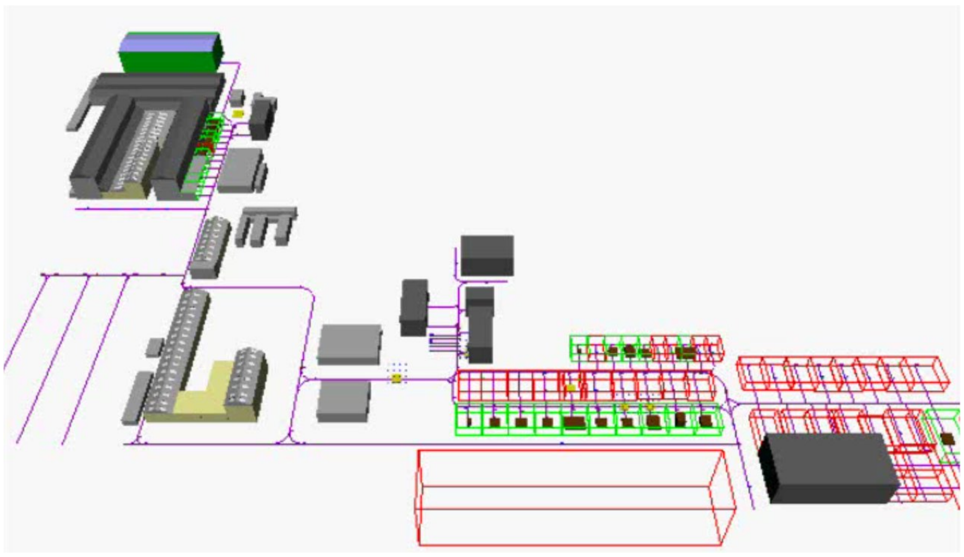


Figure 16.4: The AutoMod simulation model of OSS

We fulfilled the functional requirements to the simulation model, which were stated in section 10.1:

1. We can create a ship block, which has the following attributes

blockNo	weight	length	breadth	height	grandBlockNo	family

2. We can place a created ship block(loads) in any location(AutoMod queue) we choose.

3. We can remove a ship block from the simulation.

4. We have implemented processes that secures that ship blocks placed in AutoMod queues remains there until they are removed from the model, or picked up by a vehicle.

5. We have designed a path movement system identical to the road system in the OSS problem domain.

6. A vehicle can travel on the path movement system to a specified location(AutoMod control point) in the model, and can pick up ship blocks (load) and place ship blocks in AutoMod queues.

7. We have full control of a vehicle in AutoMod, meaning that we can navigate it to any location(AutoMod Control Point) we choose.

8. We have constructed the simulation such that it is possible to control it externally using Active-X.

A limitation with AutoMod have been that the road system at OSS mainly can be designed with the AutoMod path mover system, wherein roads do not have a breadth. The functionality of the path mover system has limited the degree of fredom in respect to modeling the passages at OSS in detail, i.e. we can not use a single road so that two KAMAG vehicles can pass each other simultaniously, without colliding.

Another limitation with AutoMod is the set of data structures it provides to developers. For instance, when a vehicle in AutoMod arrives at a queue, wherein it must place a ship block, it can retrieve the queue name of that queue, but the problem is that the queue is controlled by a process, thus we can not place the ship block in the queue without sending it to the process. We are able to retrieve the process name, but are not able to retrieve the process pointer. We solve this problem by retrieving the process pointer through the AutoMod ActiveX runtime object. An obvious solution would be for Brooks Software, which have developed AutoMod, to implement accessable hashtable datastructures in AutoMod, which would improve AutoMods functionality considerably.

16.3 DECAF

We have build a MAS with the DECAF framework and tested it with the simulation model(se section 15.2). The disadvantage of the framework where that every agent had to run in its own JVM, which made it impossible for exhanging or sharing Java objects between the agents. Everything had to go through KQML messages, which can only contain text strings.

We also modified the DECAF framework, because it is built in such a way that all the implemented agents and their belonging classes has to be placed at the default package, where the DECAF framework with its 55 classes also reside. This modification is described in 14.3.1.

The DECAF framework contains a Plan Editor which makes it quite simple to create new agent plans. The user of the framework, places tasks,actions, non-local tasks, etc, and interconnect those by line. The user can then auto generate the java skeleton files, where he needs to implement the logic of the tasks.

The agents that we implemented in DECAF can be categorized as proactive agents. Because they don't perceive their environment, when they drive to fulfill their transportation objective.

One of the main reasons for chosing this framework in this thesis was that it contained a GPGP module. But later we found, that this module was under development, so therefore we implemented our own simple planning of the transportations requests in the D-planner agent.

16.4 Cougaar

We have designed a multi-agent system in Cougaar wherein the agents have been organized into three groups, PA, CA and KVA, and the agent-to-agent relations have been implemented with the customer/provider relationship.
The KVAs[6] are more reactive than the corresponding KAMAG agents[7] we designed to the Decaf framework. In fact the KVAs can now be categorized as cognitive agents without the learning ability. The problem was that KAMAG agents did not respond to changes in the environment, such as ship blocks and other KAMAG agents, meaning that a KAMAG agent could drive through a ship block and through another KAMAG agent. This was of course not acceptable, thus we solved this problem by designing and implementing a graph representation of the OSS domain, and a path-finder algorithm based on the A* algorithm, which finds the shortest path avoiding obstacles(ship blocks). When a KVA places a ship blocks it updates the graph, by increasing the cost on edges connected to the node(location), and all KVAs are notified that the environment has changed, likewise when a ship block is picked up by a KVA, the cost of the edges are descreased, and the environment updated.

[6]KAMAG Vehicle Agents, Cougaar agents
[7]Decaf agents

The KVA is responsible for planning its own route in order to arraive at a given destination, thus implying proactiveness. It respons to changes in the enrionment, by negotiating with other KVAs(thus more social), an replanning its route when neccesary. The KVAs now place Bids on TransportTaks looking at its daily schedule to see if it can handle a transport task.

The CA is also a cognitive agent, which reasons about what agent is best to choose in a given situation. As changes occur in the environment, the CA dynamically replans its coordinations of transportation tasks, delegating tasks to the KVA with the best bid, which is estimated using a scoring function, which gives transportation tasks with the highest priority, the highest score, which maximum can be 1. The score of a bid regarding a task with priority 2, can at most be 0.5, which is obtained if the agent can pick the ship block up, exactly on time. When KVA routes intersect they will have to communicate in order to resolve the intersection problem, which contributes to the duration of both KVAs, thus another task might suffer from this conflict, requirering replanning from the CA. When a given task is considered not to be possible to solve within the deadline, the CA reports a failure back to the PA.

The PA is a purely reactive agents, when only reacts on the input from the user, by sending transportation requests, and by visualizing results regarding allocated transportation to the human user. When a task fails it is reported to the PA, which updates the results in a monitor GUI, from where the human user can accept the failure or replan his tasks. The final decision regarding replanning is therefore left up to the top-level user.

External access from the multi-agent system have been implemented as a plugin in a KVA, meaning that instead of having a centralized agent for communication as the MAS we designed with Decaf, communication is now decentralized, and only agents with the SimulationCommunication plugin have the ability to exchange data with the simulation model.

16.5 Middleware

We have designed and implemented a middleware application that makes it possible to exchange data between the simulation model and both multi-agent systems. AutoMod provides two main types of external communication: socket and ActiveX. We chose to use ActiveX because it provided extended access to AutoMod entities, such as pointers(Variables pointing at AutoMod entities).

16.6 The learning process

In this thesis we have researched a real-world problem domain, worked with several new technologies, such as the AutoMod simulation environment, the multi-agent terminology, corncerning BDI, AEIO and agents in general. Furthermore we have experimented with two different MAS frameworks, more specifically the Decaf framework involving the TAEMS language for structuring tasks in MAS, and the Cougaar framework involving the Cougaar methology. Moreover we have used the programming language C# to develope our middleware application.

Experimenting with all those new technologies and concepts have been a very educating experience, and a very comprehensive project.

16.7 Future Work

Allthough the implementation of the multi-agent systems in Decaf and Cougaar demonstrate the essential functionality, some of the functionality described in the design have not been fully implemented. Thus future work will include a full implementation of the design.

Today KAMAG vehicle drivers communicate with D-planners and truck drivers by using walkie-talkies. Truck vehicles are used to navigate the KAMAG vehicles at OSS. If the KAMAG vehicles were equipped with GPS, it would be possible to track them, and thereby using a system, like the one we have designed in this thesis, to navigate them. Thus in the future it could be possible to make the transportations at OSS fully automated.

Every morning C-planners bi-cycles around their local areas at OSS in order to check the storage locations to see what ship blocks are placed where, and what storage locations are free, before they can make their individual daily C-plans. At OSS they have considered to use the RFID[8] technology to keep track of inventory, but due to the large amount of steel used at OSS, the radio frequency signals are weakened so much, that this technology is unusable. If it were possible to apply another technology to keep track on inventory, then it would also be possible to represent the real-world locations of ship blocks in a simulation model like the one we designed. This would make it possible to synchronize the real-world environment at OSS and the simulation model, thereby updating the system and giving planners and coordinators at OSS the ability to detect, when ship blocks are misplaced, and to follow the entire flow of any ship block at OSS.

[8]Radio Frequency ID

Conclusion

The major objective with this thesis project was to design a decision support tool for coordination of the daily transportations of ship blocks at OSS, based on emulation and multi-agent technology. The major objective was categorized into three objectives in section 2.5.1. We conclude the work done by answering those objectives:

1. **Emulation:** We have designed and implemented a detailed three dimensional graphical simulation model in AutoMod. The simulation model is a scaled version of the OSS domain, which contains ship blocks, storage locations, buildings, KAMAG vehicles and a path system. Furthermore functions are implemented, such that the simulation model can be accessed by an external system for emulation purposes.

2. **Multi-agent system:** We have designed two multi-agent systems with the Decaf and the Cougaar framework respectively. The MAS designed with Decaf, only concerns a subpart of the logistic problem at OSS, but were fully implemented. The MAS designed with Cougaar fully covers the logistic planning, but were partly implemented.

3. **Middleware:** We have designed and implemented a middleware application that makes it possible to exchange data between the simulation model and both multi-agent systems. Furthermore the middleware application is able to update/insert OSS entities into our OSS-database. Moreover the middleware contains a feeder that can be used independently of the multi-agent systems to test various "what-if" scenarious in the simulation model.

Ali Cevirici Henrik Møller-Madsen

May 11, 2007

Part VI

Appendix

Appendix A

Glossary

A.1 Acronyms

ACL Agent Communication Language

ALP Advanced Logistics Project

AMS Agent Management System

ANS Agent Name Server

API Application Programming Interface

DARPA The Defense Advanced Research Projects Agency

DBMS Database Management System

DF Directory Facilitator

DVMT Distributed Vehicle Monitoring Testbed

FIPA Foundation for Intelligent Physical Agents

FIPA-ACL Foundation for Intelligent Physical Agents - Agent Communication Language

FP Feasibility Precondition. The necessary condition which has to be fulfilled before a FIPA agent can use a Communicative Act

GM General Motors

GPGP Generalized Partial Global Planning

GUI Graphical User Interface

HTN Hierarchical Task Network. The tree structure kind that is used by the DECAF tasks.

KAMAG Karlsdorfer Maschinenbaugesellschaft. A vehicle to transport one or more ship blocks

KQML Knowledge Query and Manipulation Language

MAS Multi-Agent System

MAES Multi-Agent System and Emulation

OSS Odense Steel Shipyard

PGP Partial Global Planning

PRS Procedural Reasoning System

RE Rational Effect. The effect, a FIPA agent expects, that will occur when it uses a Communicative Act.

SL Semantic Language

SQl Standard Query Language

TAEMS A Framework for Task Analysis, Environment Modeling, and Simulation

UI User Interface

VKB Virtuel Knowledge Base. Each agent has some knowledge which are located in its Virtual Knowledge Base. Virtual, because the knowledge doesn't need to reside in the agent, but can for example reside in a database.

VM Virtual Machine

A.2 English-Danish Translations

Gantry crane Portalkran

Foremen/Supervisors Værkfører

Bends Bukke

Beds Senge: Transportation vehicles to smaller steel constructions

Gravel Grus

Keel of the vessel Køllægning af skib

Girders	Drager
Kerb	Kantsten
Equipment quay	Udrustnings kaj
Corrosion protection	Rust beskyttelse
Fiery and Plasma glow	Flamme- og plasmaskæring
Rolling	Valsning
Area X	Plan X området på OSS.
Scaffold	Stillads

Appendix B

Research Phase

This chapter contains miscellaneous information collected from the research phase, which is not of major relevance to the main report.

B.1 Aerial Overview of OSS

In this section we give an aerial overview of the OSS domain(seen in figure B.1), wherein we have marked areas, and halls. The aerial photo is not up to date, thus some buildings and areas are missing, but our marks still indicates where they are currently placed.

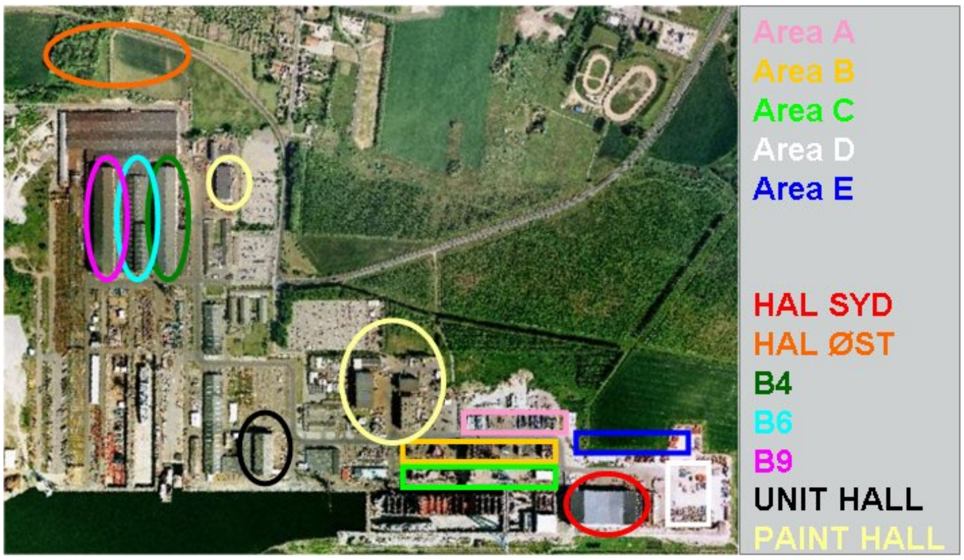


Figure B.1: Aerial overview of OSS. Areas are marked with squares and halls with ellipses

B.2 Miscellaneous problems

Following problems have been statet by the D-Planner Claus Rønaa in our discussion about the logistic problem at OSS. These problems are not considered to be within the scope of this thesis project, thus we merely state the problems:

Problems

1. The so called "Beds" are sometimes transported back and forwarded to the same place around track 6
2. If it is raining or has rained, KAMAG transporters can not access the area B1250 caused by the gravel based foundation.
3. A few weeks prior to keel of the vessel, there are an insufficient amount of girders and bends.

The following domain optimazation proposals have been suggested to solve the problems:
Optimization proposal

- Scaffolds placed at location V122-V125 could with advantage be moved to area A, a good reason to do this is that the weight of the scaffolds are limited.
- The area B1250 should be asphalted, and the kerb should be removed.
- An extension of the area east of B1250, south of B1251 and B1252 would make an faster passage of KAMAG vehicles from Hal Øst

B.3 Production Flow

This section describes the production flow at OSS, which we have adapted from their website[1]. We refer to figure 2.6 for a visual representation of the following production flow:

1. Approximately 10.000 - 12.000 ton steel pass through each month. Steel plates and profiles are being sandblasted, then painted to provide corrosion protection, before entering the ship building activity.
2. About 100.000 steel elements are cut to each ship by fiery and plasma glow. This work is done at a profile factory with a profile cutting robot and line to production of T-profiles.

[1]http://www.oss.dk

3. At this phase the building and assembling of panels takes place, which results in rectangular/right-angled sections. The process is mainly automatized welding with robots. Rolling and folding of steel plates and building of curved panels are mainly done by manual welding.

4. Assembling of rectangular sections (maximum 32m x 22m x 12m) is done in Hall 1, which are welded by offline programmable robots. Each week approximately 8.000 meters are welded. In Hall 2, bulb- and boss sections and other sections considered difficult to weld are assembled.

5. Seven painting halls are used for sandblasting and painting sections. Approximately 1000 tons paint is used every year.

6. Both large and small components are produced e.g. approximately 10.000 pipes per ship.

7. Sections of engine rooms are being equipped in the Unit hall, likewise outdoor equipments of sections in several levels takes place at the dock area.

8. Sections are assembled to large establishment blocks in the block assembly hall and by levels at the dock area.

9. OSS has got 3 docks, of which 2 of them are old ones. Dock 1 and Dock 2 are the old ones with a dimension of 300m x 44,5m x 10m. These old docks are used for storage and production of smaller parts of the ships. The third dock has got a dimension of 415m x 90m x 11m, and is used for new ships at the shipbuilding yard. The large sized blocks are established to form a ship on 6 to 10 weeks depending on the size of the ship. The gantry crane has got a payload of 1000 ton serves dock 3 and the level at the dock area.

10. The ship is equipped and systems are tested for approximately 5 weeks.

11. Finally the ship goes for a test run to Skagerrak with a subsequent finishing at Århus or Gøteborg, which takes approximately 8 days.

Appendix C

Example of a C-plan Dayreport

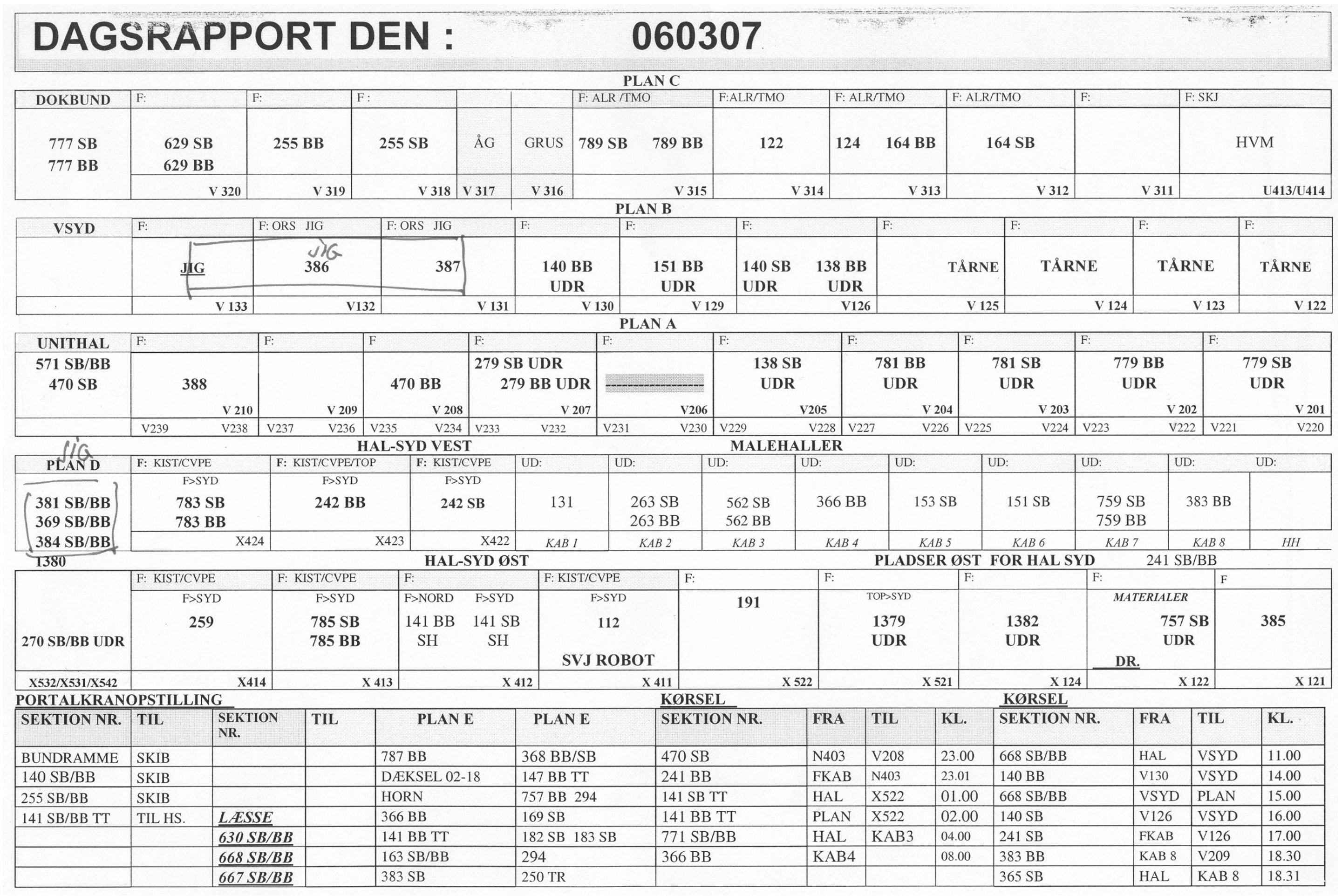
In this chapter a description of a C-plan Dayreport will be given. An example of a dayreport is given in figure C.1.The actual size of the dayreport is in A4. The date of the dayreport is given at the top of the paper and has the format *yymmdd*. The next 5 rows contains following areas and ship-blocks currently placed in those areas.

1. PLAN C
2. PLAN B
3. PLAN A
4. PLAN D, HAL-SYD VEST, MALEHALLER
5. PLADSER SYD FOR HAL-SYD, HAL-SYD ØST, PLADSER ØST FOR HAL-SYD

Each of these areas is split in places, e.g. PLAN C consists of the places V320, V319, V318, etc. Every ship-section has a ID-number and it is seen in the figure that the ship-section 255 BB[1] is placed in PLAN C at place V319.

The last part of the dayreport describes the transport and lifts to the gantry crane. The transport part contains following information: the ID-number of the section to transport, where to transport the section from/to, the transport time.

[1]BB: port side and SB: starboard



DAGSRAPPORT DEN : 060307

PLAN C

DOKBUND	F:	F:	F :			F: ALR /TMO	F:ALR/TMO	F: ALR/TMO	F: ALR/TMO	F:	F: SKJ
777 SB 777 BB	629 SB 629 BB	255 BB	255 SB	ÅG	GRUS	789 SB 789 BB	122	124 164 BB	164 SB		HVM
	V 320	V 319	V 318	V 317	V 316	V 315	V 314	V 313	V 312	V 311	U413/U414

PLAN B

VSYD	F:	F: ORS JIG	F: ORS JIG	F:	F:	F:	F:	F:	F:	F:
	JIG	JIG 386	387	140 BB UDR	151 BB UDR	140 SB UDR 138 BB UDR	TÅRNE	TÅRNE	TÅRNE	TÅRNE
	V 133	V132	V 131	V 130	V 129	V126	V 125	V 124	V 123	V 122

PLAN A

UNITHAL	F:	F:	F	F:	F:	F:	F:	F:	F:	F:
571 SB/BB 470 SB	388		470 BB	279 SB UDR 279 BB UDR	----------------	138 SB UDR	781 BB UDR	781 SB UDR	779 BB UDR	779 SB UDR
	V 210	V 209	V 208	V 207	V206	V205	V 204	V 203	V 202	V 201
	V239 V238	V237 V236	V235 V234	V233 V232	V231 V230	V229 V228	V227 V226	V225 V224	V223 V222	V221 V220

HAL-SYD VEST MALEHALLER

JIG PLAN D	F: KIST/CVPE	F: KIST/CVPE/TOP	F: KIST/CVPE	UD:	UD:	UD:	UD:	UD:	UD:	UD:	UD:	UD:
381 SB/BB 369 SB/BB 384 SB/BB	F>SYD 783 SB 783 BB	F>SYD 242 BB	F>SYD 242 SB	131	263 SB 263 BB	562 SB 562 BB	366 BB	153 SB	151 SB	759 SB 759 BB	383 BB	
	X424	X423	X422	*KAB 1*	*KAB 2*	*KAB 3*	*KAB 4*	*KAB 5*	*KAB 6*	*KAB 7*	*KAB 8*	*HH*

1380

HAL-SYD ØST PLADSER ØST FOR HAL SYD 241 SB/BB

	F: KIST/CVPE	F: KIST/CVPE	F:	F: KIST/CVPE	F:	F:	F:	F:	F
270 SB/BB UDR	F>SYD 259	F>SYD 785 SB 785 BB	F>NORD 141 BB SH F>SYD 141 SB SH	F>SYD 112 SVJ ROBOT	191	TOP>SYD 1379 UDR	1382 UDR	*MATERIALER* 757 SB UDR DR.	385
X532/X531/X542	X414	X 413	X 412	X 411	X 522	X 521	X 124	X 122	X 121

PORTALKRANOPSTILLING KØRSEL KØRSEL

SEKTION NR.	TIL	SEKTION NR.	TIL	PLAN E	PLAN E	SEKTION NR.	FRA	TIL	KL.	SEKTION NR.	FRA	TIL	KL.
BUNDRAMME	SKIB			787 BB	368 BB/SB	470 SB	N403	V208	23.00	668 SB/BB	HAL	VSYD	11.00
140 SB/BB	SKIB			DÆKSEL 02-18	147 BB TT	241 BB	FKAB	N403	23.01	140 BB	V130	VSYD	14.00
255 SB/BB	SKIB			HORN	757 BB 294	141 SB TT	HAL	X522	01.00	668 SB/BB	VSYD	PLAN	15.00
141 SB/BB TT	TIL HS.	***LÆSSE***		366 BB	169 SB	141 BB TT	PLAN	X522	02.00	140 SB	V126	VSYD	16.00
		630 SB/BB		141 BB TT	182 SB 183 SB	771 SB/BB	HAL	KAB3	04.00	241 SB	FKAB	V126	17.00
		668 SB/BB		163 SB/BB	294	366 BB	KAB4		08.00	383 BB	KAB 8	V209	18.30
		667 SB/BB		383 SB	250 TR					365 SB	HAL	KAB 8	18.31

Figure C.1: dayreport

Appendix D

DECAF

In this chapter we will describe the DECAF framework in more detail. First we will give an overview of DECAF then describe the achitecture of DECAF and at last describe the Plan Editor in DECAF.

D.1 An overview

Each agent in DECAF runs in its own JVM and has its own plan file which defines its capabilities. Two or more agents can interact with the use of KQML and only if they have registered with the ANS. The ANS is like a "white pages", where agents register themselves so other agents can find them to communicate with them. In figure D.1 the arrangement of the ANS and the agents can be seen.

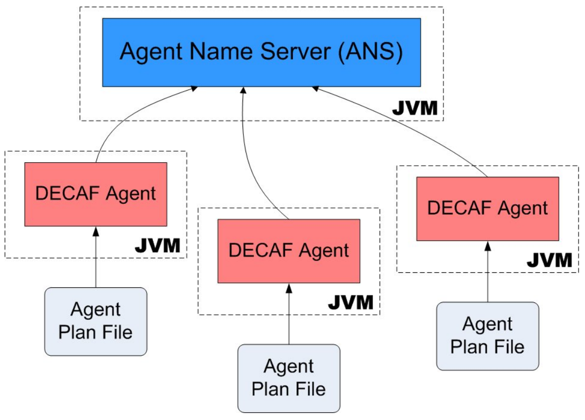


Figure D.1: DECAF overview

The capabilities of an agent are described in the plan file, which will be further explained in the Plan Editor section.

D.2 The architecture

The architecture of decaf was previously shown in figure 5.13 and is shown again in figure D.2 for convinience.

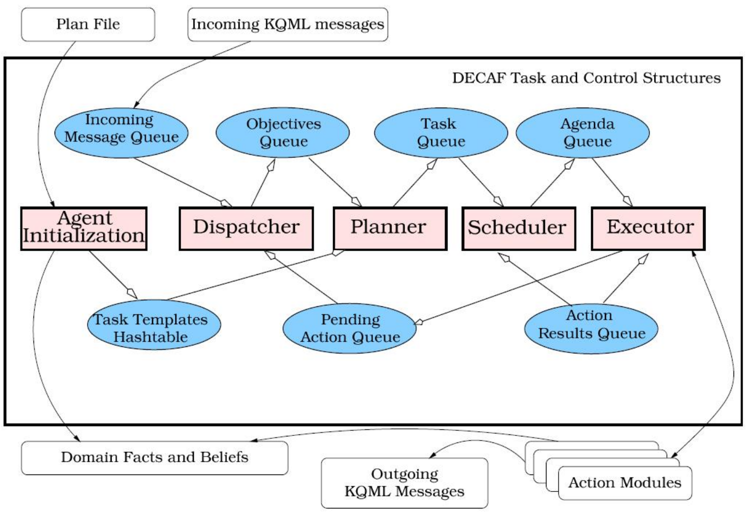


Figure D.2: Decaf architecture

There are five internal execution modules (square boxes) an the seven associated data structures (blue boxes) defined in the DECAF structure [42]. We will now give a description of these elements in the architecture with the help of [27].

- $\boxed{\textbf{Agent initialization}}$ When the agent starts it will read the Plan File and add the Tasks from that file to the *Task Templates Hashtable*. The agent might also initialize some *Domain Facts and Beliefs* which can be used in future execution of the agent. Optionally the agent might have a Startup task, from which the agent can achieve initial goals (e.g. we use Startup tasks to initialize GUI windows for different agents). Finally the agent sets up socket and network communication and registers itself with the ANS.

- $\boxed{\textbf{Dispatcher}}$ The dispatcher waits for incoming KQML messages into the *Incoming Message Queue*. When the dispatcher receives a message then it does one of following tree cases

  1. If the message is part of an already started and ongoing conversation, the dispatcher will find the action related to this conversation in the *Pending Action Queue* and continue to complete the task.

2. If the message is a start of a new conversation, then the dispatcher will create a new *objective* and place it in the Objectives Queue.
3. If the message is malformed then the dispatcher has the responsibility to send an *error message* to the sender of the KQML message.

- $\boxed{\textbf{Planner}}$ The planner waits for an *objective* in the *Objectives Queue* and when an *objective* appears then the planner will create a *TaskCell* and place it in the *Task Queue*. Currently the tasks that an DECAF agent can execute, has to be fully specified in the Plan File; but in a future release, only some parts of a task need to be specified, and then the planner will be able to search for components to fill in the missing parts of the task. Therefore in the current version of DECAF, there is not much difference between *TaskCell* and an *objective*.

- $\boxed{\textbf{Scheduler}}$ The scheduler waits until there is placed a *TaskCell* into the *Task Queue*. The scheduler will then examine the actions defined in the *TaskCell* which has the HTN form, and specify a specific execution order for these actions and place the ordered actions in the *Agenda Queue*.

- $\boxed{\textbf{Executor}}$ The executor will execute actions placed in the *Agenda Queue* if they are enabled for execution and their execution time lies between the defined "start time" and "deadline". The result of the execution will be placed into the *Action Results Queue*. The scheduler will then check whether the newly inserted "action result" is awaited from an action in the *Agenda Queue*; if so and the action doesn't miss any other results, then the action will be enabled for execution and so on.

Next we give a description of the seven data structures in the DECAF architecture.

- **Task Templates Hashtable:** The tasks defined in the Plan File for the agent, are stored as *Task Templates* in the *Task Templates Hashtable* at agent initialization. Each task template contains:
  - A name for the task
  - A list of the subtasks and actions
  - A list of inputs
  - A list of outcomes
  - An Utility function

- **Incoming Message Queue:** Contains KQML messages that are received but not yet processed by the *Dispatcher*.
- **Objectives Queue:** Contains *objectives*, which is similar to *Task Templates*, but with additional information added to the tasks, namely
  - Inputs to the *Task Template* are supplied from received KQML message
  - Scheduling information is added
  - A message ID is added in case that the message sending agent is waiting for a reply in the ongoing conversation.
- **Pending Action Queue:** Contains actions that are part of and ongoing conversation.
- **Task Queue:** Contains *TaskCell*'s, which are tasks that are ready to be executed, though they may require results from other actions that are/will be executed, before the scheduler will set the *TaskCell* to execution.
- **Agenda Queue:** Contains actions that are scheduled for execution. Each action will be executed within its defined "start time" and "deadline" and only if the enablement flag is set.
- **Action Result Queue:** Contains already executed actions with a result and an task ID, identifying which task this action belongs to.

D.3 Plan Editor

The Plan Editor is a GUI editor, used to define the capabilities of DECAF agents. The capabilities of an agent is defined in its plan file which is generated by the use of the Plan Editor. The plan file consists of following components:

- **Tasks** that the agent are able to achieve. When an agent receives a KQML message it reacts by executing a Task. The name of the task consists of an ontology and a name concatenated with and "_" inbetween.
- **Actions and subtasks** are the components beneath the Task, and they make up the Task.
- **Relationships** interconnects the Task, subtasks and actions, and defines their ordering.
- **Inputs** are parameters or provision which are passed through from incomming messages.

- **Outcomes** A Task or an action has a set of possible outcomes (a kind of "return values"). At the end of the Task/action execution only and only one of these outcomes will be returned. If a Task/action has more than one possible outcome, then the actual outcome depends on the code for the Task/action along with the input parameters.

Each of these components are shown in figure D.3, which shows an example of a Task created in the Plan Editor. In this figure there is a Task called

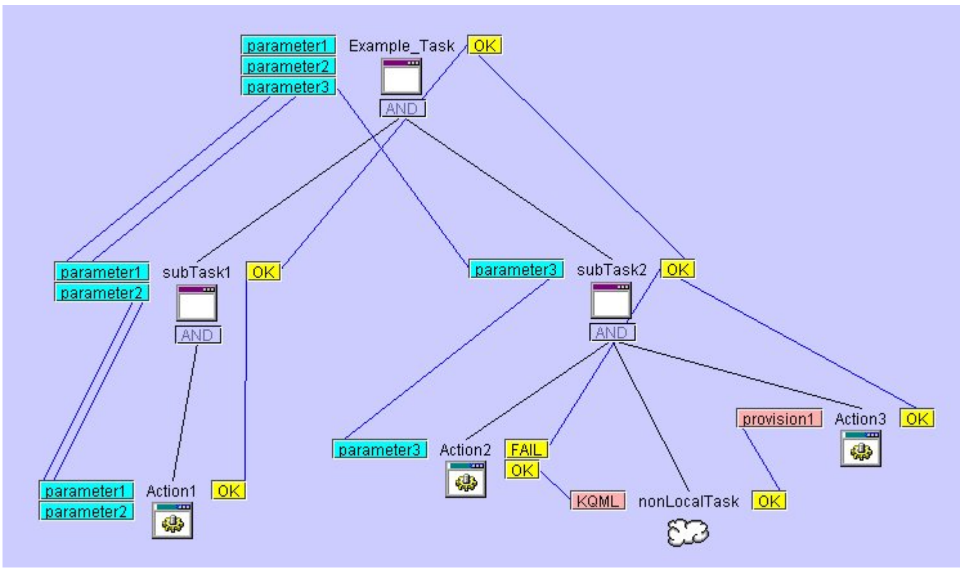


Figure D.3: DECAF Task example

Example_Task(the ontology is Example). This task has three input parameters, one outcome and two subtasks. The "AND" in the box just beneath the Task icon defines the "Characteristic Accumulation Function" (CAF) for the task. The possible values for CAF are

1. **AND** require that all sub-tasks be completed before the task completes
2. **OR** require that at least one sub-task be completed before the task completes
3. **XOR** require that at most one sub-task be completed before the task completes
4. **SUM** chooses sub tasks so that the maximum value is attained

parameter1 and parameter2 are used by Action1, parameter3 is used by Action2. Action2 has two possible outcomes, FAIL or OK and if the outcome becomes OK, then "Agent A" executing this Task will send a KQML message to "Agent B" and wait for an answer. When "Agent B" responds, "Agent A" will receive the answer with a provision and execute Action3.

In the the next subsection we will give a short description of how to use the Plan Editor application to create tasks for DECAF agents.

D.3.1 Starting, Editing and Generating

The planeditor is a JAVA application and can be started in the command prompt whith the following command[1].

```
java planeditor
```

When the Plan Editor starts the GUI shown in figure D.4 will appear.

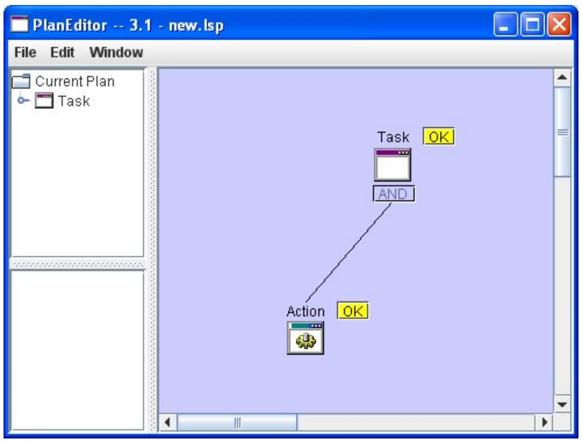


Figure D.4: DECAF Start Window

As seen from the figure, there exists one Task and one Action. It is now possible to add another tasks or action and interconnect them by lines. To add a new task/action choose Edit in the menubar and click on Add Item as shown in figure D.5; then a new GUI box will appear where it is possible to select a task or an action. To draw a line between two components, middle-click the mouse on the source component and then middle-click on the destination component.

When all the components have been drawn, the time has come to generate skeleton code for tasks/actions. First of all save the drawed components (this will be the plan file with the extension .lsp); then choose "File" in the menubar and click on "Generate Code" as shown in figure D.6.

[1]We have used eclipse and ant to run the Plan Editor. An ant build file is located in the delivered CD-ROM

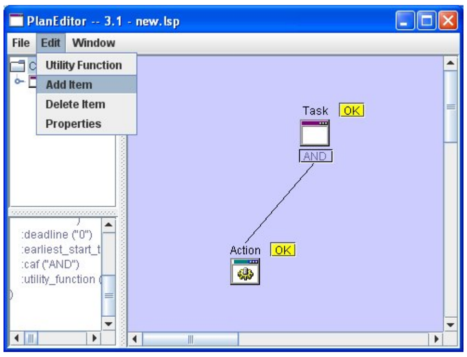


Figure D.5: Adding and item

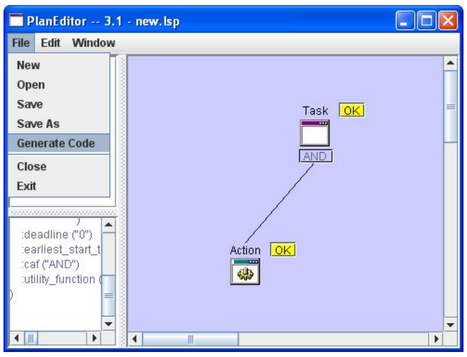


Figure D.6: Generating code

As an example let us generate code for the task shown in figure D.3. The Plan Editor will create two classes, namely subTask1.java and subTask2.java. The generated skeleton code in subTask2.java are shown beneath.

```
import decaf.Agent;
import decaf.KQMLmsg;
import decaf.LinkedListQ;
import decaf.ProvisionCell;
import decaf.Util;
import java.io.*;
import java.net.*;
import java.util.*;

public class subTask2
{
  public subTask2()
  {
    System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
    System.out.println("% subTask2 Zero constructor %
        <==============");
    System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
  }

  public ProvisionCell Action2(LinkedListQ Plist, Agent Local)
  {
    String message = new String(Util.getValue(Plist, "MESSAGE"));
    String parameter3 = new String(Util.getValue(Plist, "
        parameter3"));

    if()
    {return new ProvisionCell("SOME STRING HERE","FAIL");}

    else if()
    {return new ProvisionCell("SOME STRING HERE","OK");}
  }
  public ProvisionCell Action3(LinkedListQ Plist, Agent Local)
  {
    String message = new String(Util.getValue(Plist, "MESSAGE"));

    return new ProvisionCell("SOME STRING HERE","OK");
  }
}
```

The two actions above have to be filled in with code, which will be executed when the agent needs to run subTask2.

For further information about the Plan Editor, construction of plans for an agent and DECAF programming, please consult [42].

Appendix E

COUGAAR

E.1 What Kind Of Problem is Suitable to a Cougaar solution?

The problems that are suitable to a Cougaar solution include the following [20]:

- Problem domains that entail hierarchical decomposition and tracking of complex tasks
- Complex application domains involving integration of distributed separate applications and data sources
- Domains involving the generation and maintenance of dynamic plans in the face of execution
- Highly parallel applications with relatively loose-coupling and low-bandwidth communications between parallel streams
- Domains too complex to model monolithically, best modeled by emergent behavior of components

E.2 Architecture

The Cougaar architecture is shown in figure E.1. The collection of all agents in a cougaar system is referred to as the community. Agents that share some common functionality are categorized into groups, called communities. A society is also modeled with a Cougaar node, which contains its own Java virtual machine. As seen in figure E.1 agents are composed of multiple plugins. Those plugins represent the agents capabilities and behaviour. Agents can request different services that are includded in the Cougaar framework. And several agent binders. We will not go into further detail regarding the overall architecture, but merely describe the most important functionalities we have used, in the following sections.

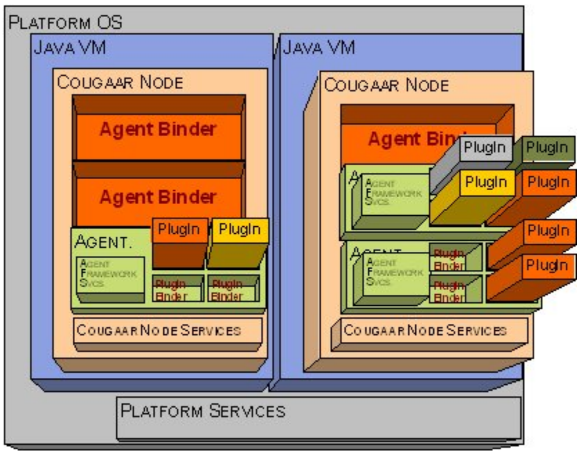


Figure E.1: Cougaar Architecture

E.3 Cougaar Agents

Agent Interaction

Agents - in the context of Cougaar, interact by sending messages through the node-level MessageTransport facilities. Agents that are part of the same society have an identical code base, and are instances of the same class, but this is not required, allowing long lived, very large societies to evolve over time.

Messages received are queuried and later passed on to a LogicProvider, that makes the appropriate changes to the blackboard or plan, though some messages will be handled instantly if neccesary.

Plugins have their own thread of execution, which are implemented with Java Thread pools, thus supporting a large set of agents whithout overhead per thread. By using notifications and alarms, agents are insured to run only when needed, reducing overhead and optimizing performance.

Timers

Every agent has a “real-time” system clock and an “execution-time” planning clock, which both have a millisecond-level of granularity.

Every agents has two timers: a "real-time" system clock and an "execution-time" planning clock, these timers can be used to allow plugins to request rescheduling at specific times.

The execution-time alarms are based on a seperate clock, and are very useful, since they run slower or faster than one second per second, making them useful for simulation purpose.

The execution-time planning clock can run continuously or as advanced time-steps. Plugins have access to this functionality through the family of "wake" methods on the standard Plugin base classes.

Alarms and timers have a millisecond-level granularity. A Plugin will be awakened as soon as posible after the alarm instant, which are constrained by load, other plugin function,etc.

Timers are completely local and are not intended for usage as synchronization between agents.

ConfigFinder

Plugins and other components, can use the ConfigFinder as a relative path to a file, and it supports "file", "http" and "ftp", and thereby a file-based search facility. The ConfigFinder is not intended for usage of general-purpose retrievement, but is appropriate for retrieving data like keys and other contact information for accesing databases, wherein general-purpose information are more appropriately stored.

E.3.1 Blackboard

The blackboard serve as local memory of an agent, wherein objects can be added/removed/changed by the agents components, and the blackboard defines an asynchronous publish/subscribe API. When agents receive messages from other agents, they can alter the content of their local blackboard.

Plugins uses a data structure to communicate with the agent and through the agent with the rest of the society. The blackboard holds various objects of interest to some or no Plugins and the agent itself. The objects on the blackboard do not know which Plugins will process it, and if the Plugin is local or placed in the other side of the globe.
A single global blackboard would be a considerable performance bottleneck, and a single point of failure.
Access to the blackboard is controlled by transactions, which behavior is

implemented with a private internal object called a Distributor, that is responsible for coordinating intercations between Plugins, persistence and the agent. Transactions are interpreted by a set of objects called LogicProviders. LogicProviders add additional business-logic behavior to changes on blackboard-level made by Plugins. LogicProviders handles messages and acts upon them, for instance by updating the blackboard.
EnvelopeLogicProviders listens to changes on the blackboard and acts upon them, for instance by sending messages to other agents.

Subscribers

Plugins are given a proxy object called a Subscriber, that handles most interactions with the blackboard. The Distributor sends Envelopes to the Subscriber, which then updates the Subscriptions with the changes or queries the changes for later processing. Changes are made while the Plugin is idle, and are queried when the Plugin is running.

Transactions

Blackboard transactions protect the consistency of the set of objects in the blackboard that are visible at a given time. Transactions can be represented as a collection of add, change and remove object messages, that should be applied to the blackboard. It is important to notice that is not the internal state of object, which are controlled by transactions, but instead the events of add, change and remove objects.

Subscriptions

Subscriptions are the only way to access Plans, and they are specified by Predicates. The entire blackboard set intializes the Subscriptions, and transaction Envelopes updates Subscriptions.

Predicates are implementations of the utility class UnaryPredicate and are executed once per object change, thus predicates should be written as short as possible. Predicate methods execute(Object) returns true if and only if the object is part of the set specified in the predicate.

Plugins can use the function getAddedCollection(), getRemovedCollection() and getChangedCollection(), to see if any object have been added, deleted or changed, which is also refered to as delta list supported by Subscriptions (IncrementalSubscription).

Subscriptions include the following classes:

- CollectionSubscription : tracks the contents and the detailed ChangeReports. Also allows specification of what sort of Collection to use internally. The internal Collection is often specified to be a class which keeps elements sorted, hashed, or otherwise arranged for the convenience of the Plugin.
- IncrementalSubscription : adds add/remove/change lists to CollectionSubscription.
- Subscription : is an abstract base class which may be used to implement any other desired behavior.

Queries

Queries are functional identical to a sequence of subscribe followed by retrievel of the results and then to unsubscribe. Plugin processing stops while querying is processed, and there are no guarantees that a query with Predicate P and a Subscription with Predicate P will match at a given time.

Distributor

The Distributors job is to coordinate the agents messages and preserve blackboard persistence.

The Distributor receives Transaction deltas (Envelopes) on the Plugin side, and is responsible for updating the blackboard, and next the Subscribers, which are interested in the changes. Messages are recieved from the agents MessageTransportClient service and forwarded to the blackboard, which then invocates LogicProviders.

E.4 Component Model

Cougaar's interface is closely modeled on the Java BeanContext API, retaining most of the terminology for Class and Method names, but implements the API without the UI-centric, which is required by BeanContext.

Cougaar adds additional layers to prevent abuse of component references, such layers include the following:

- Binders between parent and child components
- ServiceProxies between server and client objects

All interaction between between components midiates by the Binder. The Cougaar Component model is designed to function within a single Java VM, thus there are no direct support for remote component relationships.

Components can only have one parent component; a component that contain another component is called a container, and a component can have multiple child components and containers. Containers are required to implement java.util.Collections API in order to support removing, and adding of child components.

Appendix F

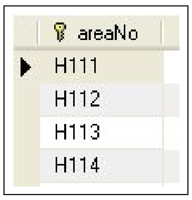
Database

In this appendix we describe the contents of the database used in our system, and how to install the MySQL database server on windows.

F.1 Tables and their content

The database contains following tables.

- **areadata** has one field areaNo, which contains all the names of the locations at OSS. This table is used by the Feeder GUI window(see figure 13.4), to extract all the location names into the drop down boxes, which are named "Location". A part of the table is shown in table F.1



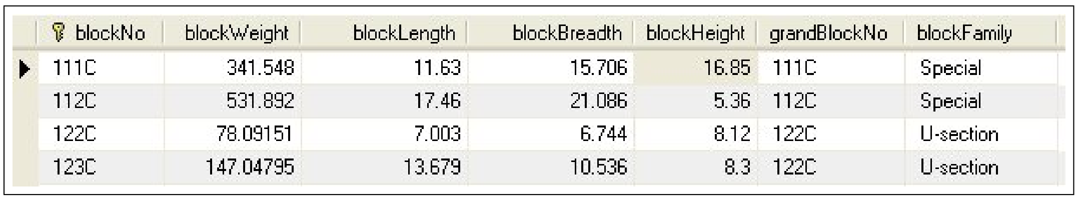
areaNo
H111
H112
H113
H114

Table F.1: A part of the areadata table

- **blockdata:** contains the every ship block for the L203 series ship build at OSS. This table has the following fields:
  - *blockNo:* The unique name of the ship block
  - *blockWeight:* The weight of the ship block measured in tons
  - *blockLength:* The length og the ship block measured in meters
  - *blockBreadth:* The breadth of the ship block measured in meters
  - *blockHeight:* The height of the ship block measured in meters

  - *blockgrandBlockNo:* The grandBlock to which this ship block is a part of.
  - *blockFamily:* The type/kind of the ship block

A part of the table is shown in table F.2

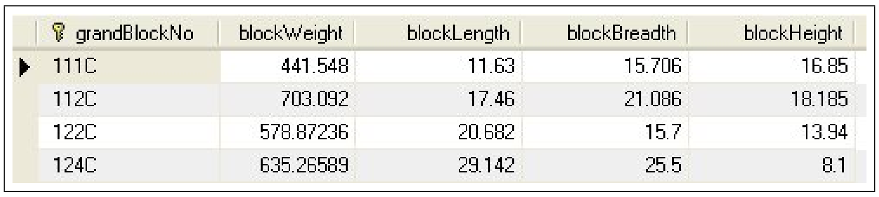


blockNo	blockWeight	blockLength	blockBreadth	blockHeight	grandBlockNo	blockFamily
111C	341.548	11.63	15.706	16.85	111C	Special
112C	531.892	17.46	21.086	5.36	112C	Special
122C	78.09151	7.003	6.744	8.12	122C	U-section
123C	147.04795	13.679	10.536	8.3	122C	U-section

Table F.2: A part of the blockdata table

- **grandblockdata** contains every grand ship block for the L203 series ships build at OSS. This table has the following fields:
  - *grandBlockNo*: The unique name of the grand ship block
  - *blockWeight:* The weight of the grand ship block measured in tons
  - *blockLength:* The length og the grand ship block measured in meters
  - *blockBreadth:* The breadth of the grand ship block measured

  A part of the table is shown in table F.3



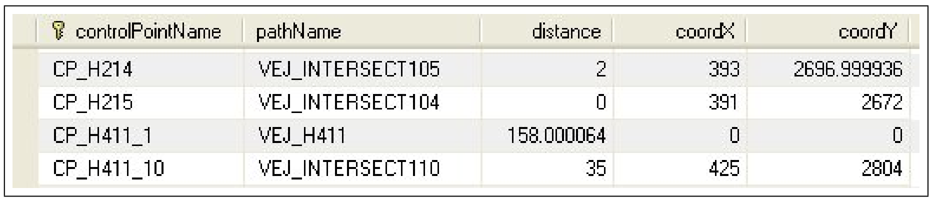
grandBlockNo	blockWeight	blockLength	blockBreadth	blockHeight
111C	441.548	11.63	15.706	16.85
112C	703.092	17.46	21.086	18.185
122C	578.87236	20.682	15.7	13.94
124C	635.26589	29.142	25.5	8.1

Table F.3: A part of the grandblockdata table

- **controlpoint** contains all the nodes shown in figure 9.4. Al the nodes in this table are related to the controlpoints in the OSS simulation model. This table has the following fields:
  - *controlPointName*: The unique name of the controlpoint in the OSS simulation model
  - *pathName:* The path in the OSS simulation model where this controlpoint is placed

  - *distance:* The placement length of the controlpoint from the start of the path measured in meters
  - *coordX:* The x-coord of the controlpoint in the OSS simulation model, measured in meters
  - *coordY:* The y-coord of the controlpoint in the OSS simulation model, measured in meters

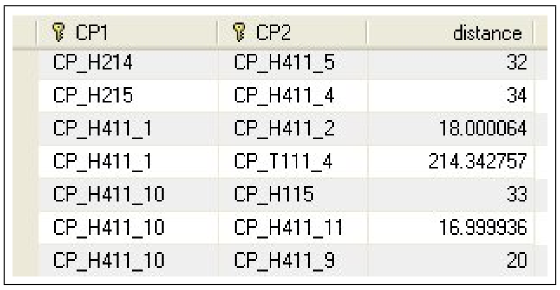
  A part of the table is shown in table F.4



controlPointName	pathName	distance	coordX	coordY
CP_H214	VEJ_INTERSECT105	2	393	2696.999936
CP_H215	VEJ_INTERSECT104	0	391	2672
CP_H411_1	VEJ_H411	158.000064	0	0
CP_H411_10	VEJ_INTERSECT110	35	425	2804

Table F.4: A part of the controlpoint table

- **distance** contains the distances between the controlpoints that are connected directly by a path, without any intermediate controlpoints. A part of the table is shown in figure F.5. From this table we can see



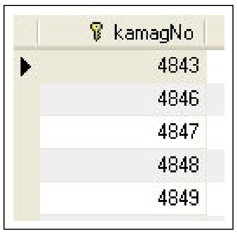
CP1	CP2	distance
CP_H214	CP_H411_5	32
CP_H215	CP_H411_4	34
CP_H411_1	CP_H411_2	18.000064
CP_H411_1	CP_T111_4	214.342757
CP_H411_10	CP_H115	33
CP_H411_10	CP_H411_11	16.999936
CP_H411_10	CP_H411_9	20

Table F.5: A part of the distance table

  that CP_H411_1 has two neighboring controlpoints (CP_H411_2 and CP_411_4), and the distances to these neighbors are also seen (18 meter and 214 meter respectively).

- **kamag** contains the KAMAG vehicles at OSS. The table and its contents are shown in table F.6. This table are used by the feeder GUI (just as the areadate table) to extract all the KAMAG vehicle No's into the drop down boxes, which are named "Kamag".

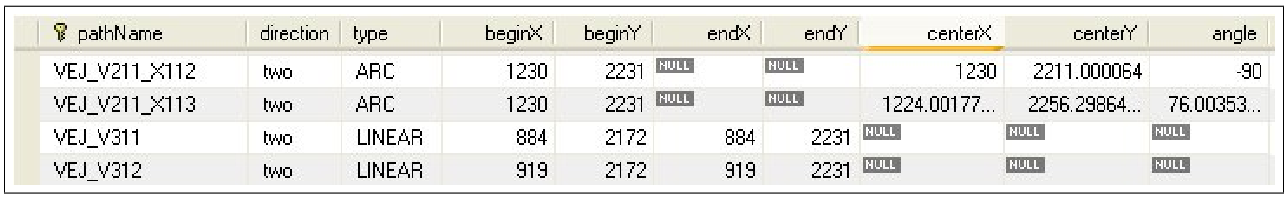
- **path** contains all the paths in the OSS simulation model. There are two types of paths, straight line paths and arc paths as described in



kamagNo
4843
4846
4847
4848
4849

Table F.6: The kamag table

section 14.2.2. Tabel F.7 shows a part of the table. In this table it is seen that the first two paths are arc paths, and that the other two paths er straight line paths.



pathName	direction	type	beginX	beginY	endX	endY	centerX	centerY	angle
VEJ_V211_X112	two	ARC	1230	2231	NULL	NULL	1230	2211.000064	-90
VEJ_V211_X113	two	ARC	1230	2231	NULL	NULL	1224.00177...	2256.29864...	76.00353...
VEJ_V311	two	LINEAR	884	2172	884	2231	NULL	NULL	NULL
VEJ_V312	two	LINEAR	919	2172	919	2231	NULL	NULL	NULL

Table F.7: A part of the path table

F.2 MySQL Server Installation Guide

In this section we will describe how to install the MySQL server software provided in the included CD-rom[1]. The software is located in the MySQL directory in the CD.

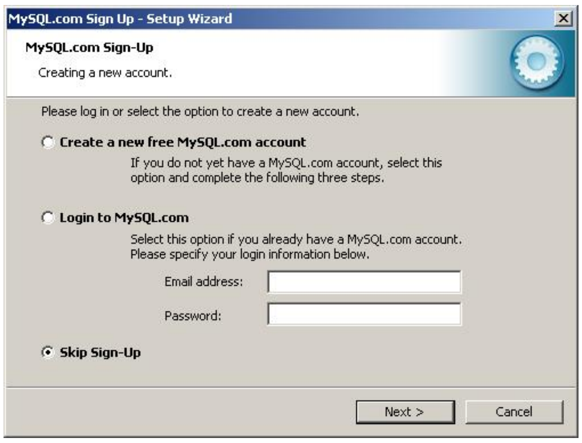
First install the MySQL server by installing the file “Setup.exe” located in the zip file “mysql-5.0.21-win32.zip”. When the following window (figure F.1) appers during installation, choose skip signup.

Choose a root password for the MySQL server, and finish the installation.

Next install the MySQL administrator, that is the file “mysql-administrator-1.1.9-win.msi”. With the use of the MySQL administrator login to the MySQL database as the root user and add the user “oss” with the password “sso”. Then add the “oss” schema and assign all available privileges to this schema.

To connect to the database from the Middleware we have to use a ODBC driver. In the following section the setup of the ODBC driver will be explained.

[1]The software can also be downloaded from MySQL’s homepage www.MySQL.org


MySQL.com Sign Up - Setup Wizard

MySQL.com Sign-Up

Creating a new account.

Please log in or select the option to create a new account.

( ) Create a new free MySQL.com account

If you do not yet have a MySQL.com account, select this option and complete the following three steps.

( ) Login to MySQL.com

Select this option if you already have a MySQL.com account. Please specify your login information below.

Email address:

Password:

[✓ Skip Sign-Up]

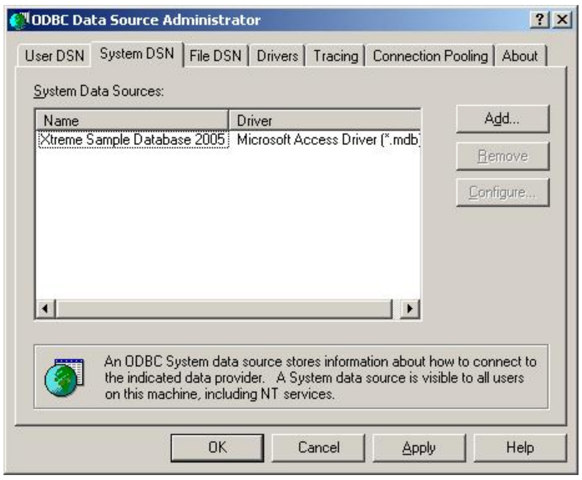
Next > | Cancel

Figure F.1: MySQL Sign Up Window

F.2.1 Setting up the MySQL ODBC Connector

Install the odbc software; that is the file "mysql-connector-odbc-3.51.12-win32.msi". To set up the MySQL ODBC Driver follow these steps:

1. Open the Control Panel | Administrative Tools | Data Sources (ODBC) window (figure F.2) and click the Add button.


ODBC Data Source Administrator

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

System Data Sources:

Name	Driver
Xtreme Sample Database 2005	Microsoft Access Driver (*.mdb

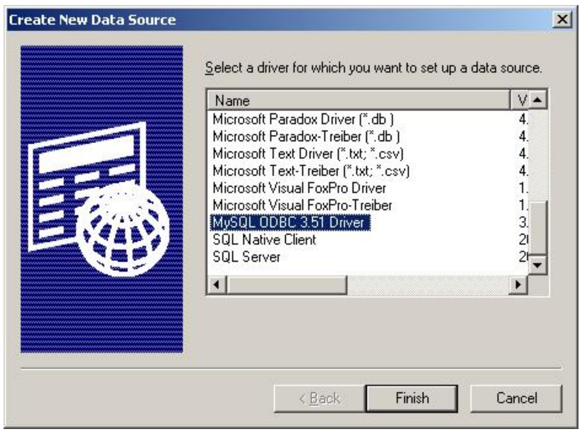
Add... | Remove | Configure...

An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

OK | Cancel | Apply | Help

Figure F.2: Data Sources Window

2. Select "MySQL ODBC 3.51 Driver" and click the Finish button (figure F.3)



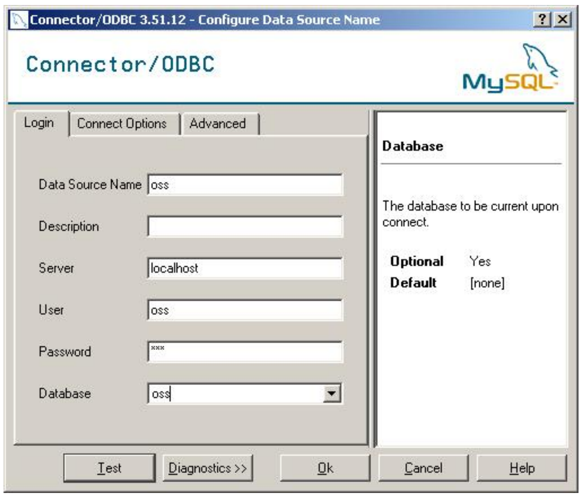
Create New Data Source

Select a driver for which you want to set up a data source.

Name	V
Microsoft Paradox Driver (*.db)	4.
Microsoft Paradox-Treiber (*.db)	4.
Microsoft Text Driver (*.txt; *.csv)	4.
Microsoft Text-Treiber (*.txt; *.csv)	4.
Microsoft Visual FoxPro Driver	1.
Microsoft Visual FoxPro-Treiber	1.
MySQL ODBC 3.51 Driver	3.
SQL Native Client	2(
SQL Server	2(

Figure F.3: Create New Data Source Window

3. In the Connector/ODBC window, enter the values as seen in figure F.4 and as password enter “sso”.



Connector/ODBC 3.51.12 - Configure Data Source Name

Connector/ODBC

Data Source Name	oss
Description	
Server	localhost
User	oss
Password	***
Database	oss

Database

The database to be current upon connect.

Optional	Yes
Default	[none]

Figure F.4: Connector/ODBC Window

4. Click the Test button and it should succeed.

F.2.2 Howto insert Ship Block Data into Database

When we asked for data for all the ship blocks for a ship we got handed out an excel file; “L203 Blockdata Rev F.xls”. We extracted the needed information for our system from that file into the the file “L203 Blocks.xls” and converted it into XML[2]. To put the information for the ship blocks into the database use the Feeder as shown in figure F.5; then choose the file "L203 Blocks.xml" and click Open. The same procedure can be followed to insert the kamags and locations into the database. The kamags are located in the file "kamags.xml", the locations in the file "locations.xml".

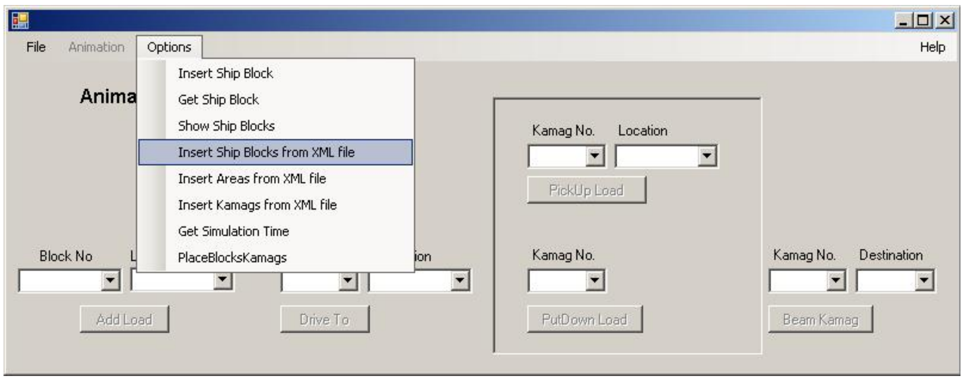


Figure F.5: Adding ship blocks from XML file

[2]All the excel and xml files are located in the OSSData directory in the CD

Appendix G

Using Automod

This Appendix serves to give a short overview on the features we have used in Automod. Here is everything one needs to know in order to build a simulation model in Automod like the one we have build. The information in this appendix only concerns how it works, but not does not describe what it is. We refer to the Automod manual for en overall describtion of Automods functionality.

G.1 Getting started

G.1.1 New project

First we should open Automod as seen in figure G.1, then we choose “New” in the “File” menu and select a name for your model.

Once a new model has been created you will see the “Process System”(figure G.2) wherein you define, create, modify, delete and use different types in Automod.

G.1.2 Saving Your Project Correctly

There are two ways of saving your project; either use “save” or “export”. Use “save” when you have temporary changes in your project, which you are not sure you want to keep; mostly for testing purpose. Use “export” when you have made permanent changes to your model, you want to keep. When you export your model, Automod saves it in the process. An exported model can be transfered to another machine, and it will be identical to the original model. If you just choose to save the model, it can not be transfered to another computer.

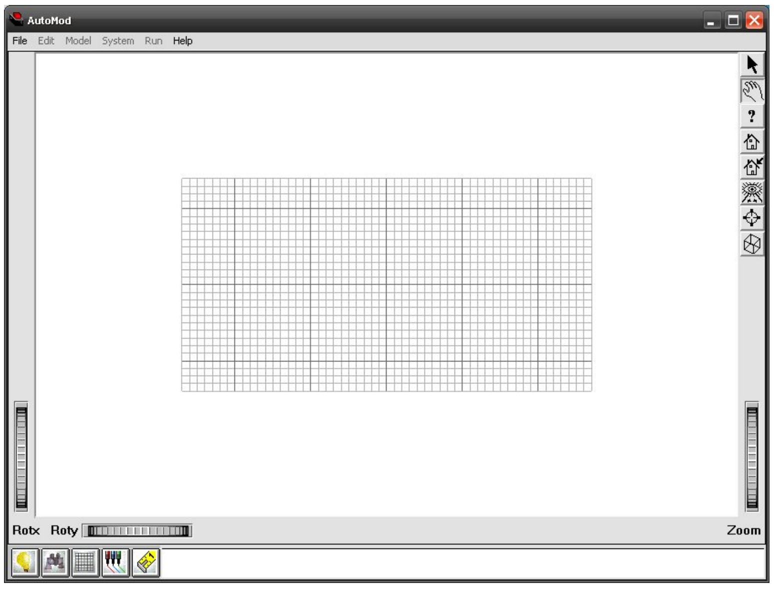


Figure G.1: Automod Environment

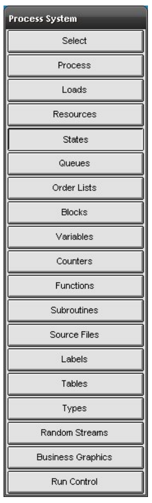


Figure G.2: Process system

G.1.3 Saved project

Returning to your previous work; you want to open your model, either you go to your "dir" archieve, which is your temporary saved project, or you go to your "arc" archieve, which is your permanent saved model. Remember to open the correct model.

G.2 Loads

Loads are dynamic elements in the simulation. We can create different load types and attach attributes to loads. Different load types helps distinguish between different product types, such as ship blocks and raw material.

Choose "Load" from the "Process System" and the window seen in figure G.3 will appear. On the left side you can define new load types, and on the right side you can define load attributes. All load types will have the same attributes, so its not like object oriented programming, where you would define a block object with some attributes, and a material object with some other attributes. The attributes are local to each load, thus changing the value for a load attribute will not change every loads attribute.
To define a new load type press the corresponding "new" button seen in figure G.3, and the window shown in figure G.4 will appear, wherein the

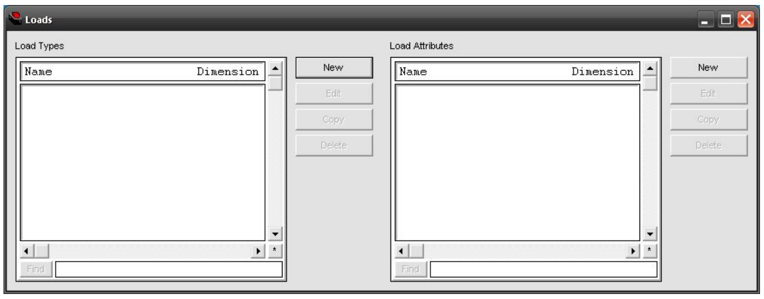
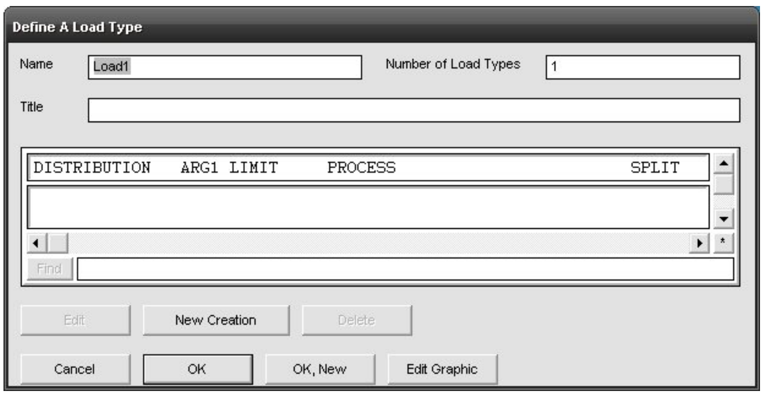


Figure G.3: The loads window

name of the type is defined, with other information such as which process will handle the load at creation. We now must press the button "New Creation", wherein we specify:

- Generation Limit : How many units of this load you want to produce
- Split :
- First Process : What process should handle the load at creation
- Distribution : Different time distribution, e.g. constant time, creates new loads after a constant amount of time.
- Mean : the time unit for load distribution, e.g. 5 seconds or 10 hours



Name: Load1
Number of Load Types: 1
Title:

Figure G.4: Defining a load type

You must choose the process which should handle the load at creation, by pressing the "First Process" button a list with all process in the system will appear. See section G.3 for information on how to design a process.

G.3 Process

A process is the virtual control or handling of loads. Every load must be in a process in order to exist. Loads enters a process, herein they execute some instructions, and finally they leave the process. A load can either be sent to another process, or sent to "die"; "die" is a standard process that kills the load, and if the programmer do not specify another process to sent the load to before it leaves, the load is automatically sent to die by Automod.

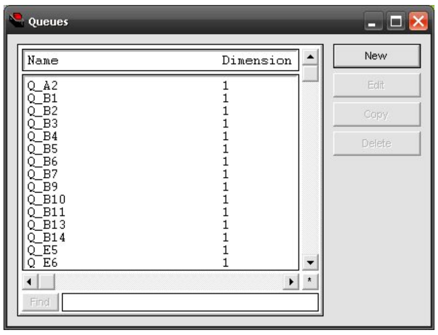
Here is an example of how to write process logic:

```
begin P_FirstGrinder arriving procedure
  move into Q_FirstGrinder_wait     /* Move into the waiting queue
      */
  move into Q_FirstGrinder          /* Move into the processing
      queue */
  get R_FirstGrinder                /* Claim the grinder */
  use R_Operator for n 20, 3 sec    /* Loading */
  wait for 4 min                    /* Grinding */
  use R_Operator for e 30 sec       /* Unloading */
  free R_FirstGrinder               /* Free the grinder */
  send to P_SecondGrinder           /* Send to the next process */
end
```

As can be seen from the code example above, we use prefix in front of names, such that "Q_", "R_", "P_", "L_" indicates a queue, resource, process or load.

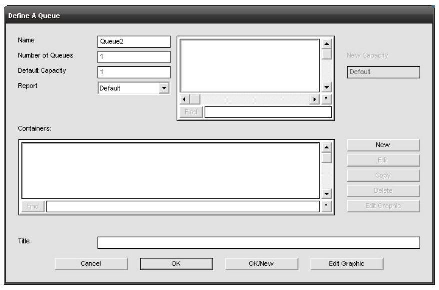
G.4 Queues

Queues are used as containers for loads, thus loads can move in and out of queues. To define a queue select the "Queues" button from the "Process System" and the Queues window seen in figure G.5 will appear, wherein every queue in your system will be listed. Now you press the "New" button and the window for creating queues appear as seen in figure G.6, where you fill in information like "Name", "Number of queues" and "Default Capacity", which is the amount of loads that can reside in the queue. If you want to design some container for your queue, e.g. if you require som special way of placing the loads in the queue, then select the "New" button shown in figure G.6, where you can chose your own cell size, and how you want the loads to be placed, e.g. in a horizontal or vertical direction.



Name	Dimension
Q_A2	1
Q_B1	1
Q_B2	1
Q_B3	1
Q_B4	1
Q_B5	1
Q_B6	1
Q_B7	1
Q_B9	1
Q_B10	1
Q_B11	1
Q_B13	1
Q_B14	1
Q_E5	1
Q_E6	1

Figure G.5: The queues window

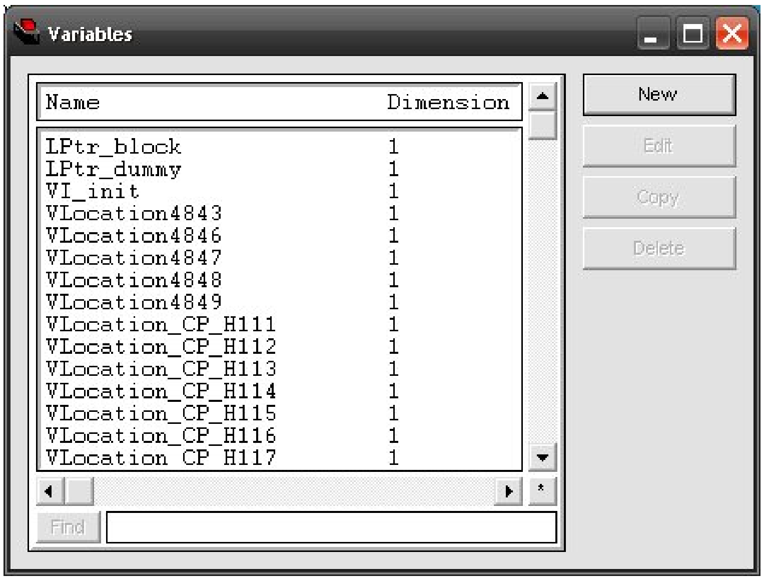


Name	Queue2
Number of Queues	1
Default Capacity	1
Report	Default
New Capacity	Default
Title	

Figure G.6: Define a queue

G.5 Variables

To store information in your simulation logic, you can take advantage of variables, which can be any type in Automod like strings, integers, Load pointers, queue pointers, or a type you define on your own. Variables in Automod are global for all objects/entities, which means that if you change this variable from one process, the value is changed for every process in your system, thus variables must be handled with great caution, when accessing a variable from multiple places, which could result in data inconsistence. For creating a variable press "Variables" in the "Process System" and the window in figure G.7 will appear.

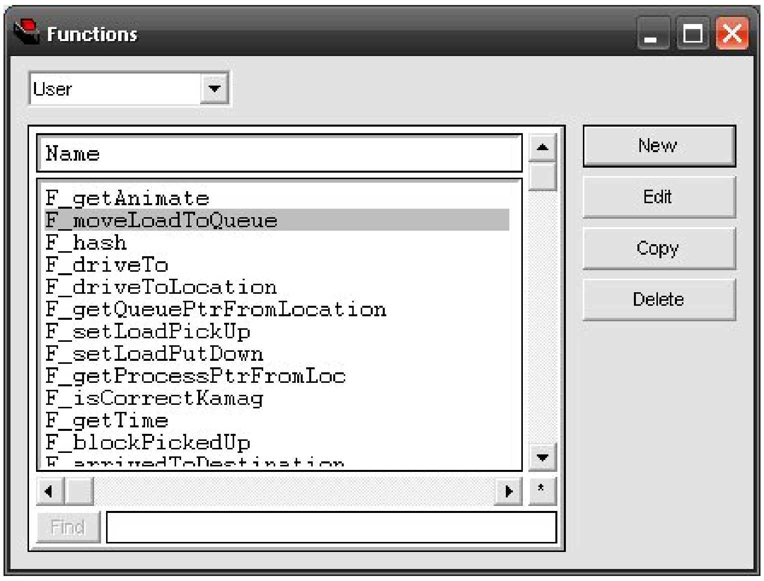


Name	Dimension
LPtr_block	1
LPtr_dummy	1
VI_init	1
VLocation4843	1
VLocation4846	1
VLocation4847	1
VLocation4848	1
VLocation4849	1
VLocation_CP_H111	1
VLocation_CP_H112	1
VLocation_CP_H113	1
VLocation_CP_H114	1
VLocation_CP_H115	1
VLocation_CP_H116	1
VLocation_CP_H117	1

Figure G.7: Variables

G.6 Functions

To make a function press "Functions" in the process system. The windows in figure G.8 will appear. There is a combo box with "user" written in it as default, which means that it shows the functions you have defined, the other fields in the combo box are build in functions, that can be used in your logic. Press "New" to create a new function and fill in the information seen in figure G.9, that is "Name", which should start with prefix "F_" during to Automod notation, the "Type" that the function should return when done executing, and you will have to define the parameters that should be passed with the function call. Remember to think about the order in which you declare the function parameters, since they define the order of parameters, when you call the function later on.



```
Name
F_getAnimate
F_moveLoadToQueue
F_hash
F_driveTo
F_driveToLocation
F_getQueuePtrFromLocation
F_setLoadPickUp
F_setLoadPutDown
F_getProcessPtrFromLoc
F_isCorrectKamag
F_getTime
F_blockPickedUp
```

Figure G.8: The function window

Once you have declared a function you will have to write the logic in it, which is done by making a source file as desribed in section G.7. If you have called your function "F_calculate" the source file name should be "F_calculate.m".

G.7 Source files

Source files is were all the fun stuf happens; if you are a programmer that is. Unfortunately Automod isn't really made for programmers, but more

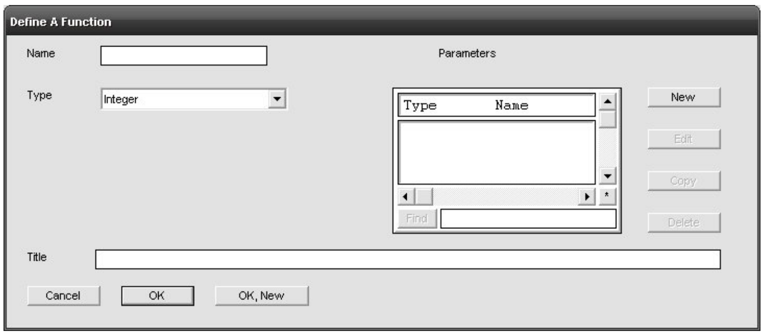
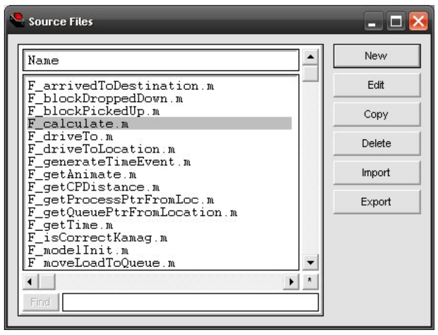


Figure G.9: Define a function

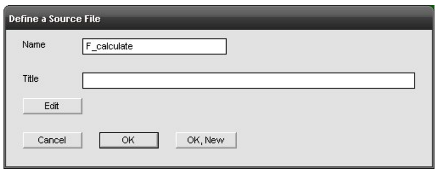
for production engineers , who don't have much experience with programming, this means that a lot of freedom has been taken from the developer, and a simple thing like declaring a new function is very difficult. You are not allowed just to write all your code in some source files and then compile them to run the program, instead you are forced to interact with the "Process System" GUI to make anything happen.

For making a new source file, press the "Source Files" button in the process system, and type the name of the source file in the box seen in figure G.11, e.g. "F_calculate.m" and remember to put the extention ".m" on the name, otherwise the function will not work.



```
F_arrivedToDestination.m
F_blockDroppedDown.m
F_blockPickedUp.m
F_calculate.m
F_driveTo.m
F_driveToLocation.m
F_generateTimeEvent.m
F_getAnimate.m
F_getCPDistance.m
F_getProcessPtrFromLoc.m
F_getQueuePtrFromLocation.m
F_getTime.m
F_isCorrectKamag.m
F_modelInit.m
F_moveLoadToQueue.m
```

Figure G.10: The source file window



Name: F_calculate

Figure G.11: Declare function name

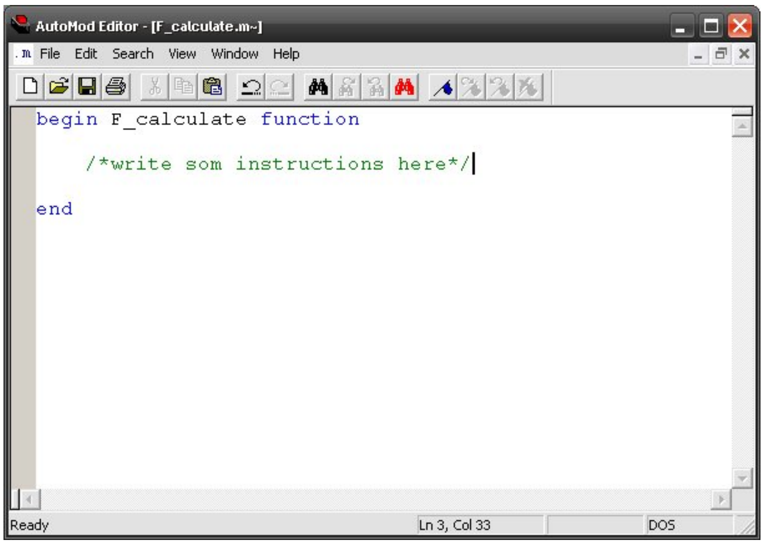
Once you have declared the name of the source file, press the "Edit" button, resulting in the opening of an editor as seen in figure G.12, wherein you write the logic for your function or process. The source file must contain the following for a function declaration:

```
begin F_calculate function
```

```
    put some instructions here

    return 1
    /* this return is required and should match the return type in
       your function declaration */

end
```


AutoMod Editor - [F_calculate.m~]
File Edit Search View Window Help

```
begin F_calculate function

    /*write som instructions here*/

end
```

Figure G.12: Automod Editor

When writing a source file for a process, you only have to specify a begin and an end command, as seen below:

```
begin
  put some instructions here
end
```

Note that is is not possible to close down the editor when there are errors, which means that can't proceed making other changes somewhere else in the Automod environment, which can be quite annoying, again limiting the developers creativity, but ensuring that no mistakes are made before closing down, can be useful for new developers.

G.8 Placing your graphics

This section guides you to placing your graphics after creating elements, such as queues, which requires visualization in your simulation model.

Select “Queue” from the process system (figure G.2), then select a queue from the queue list (figure G.5), and select “edit”

Appendix H

Automod ActiveX API

In this chapter the syntax for the Automod ActiveX methods will be shown.

H.1 CallFunction method

Purpose

Use the CallFunction method with the AutoMod runtime object to call a user-defined AutoMod function in the model during a simulation.

Syntax

```
amx.CallFunction(funcName, params)
```

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
CallFunction	The name of the method.
funcName	A string that indicates the name of the function in the AutoMod model that you want to call.
params	An array of variants that define the arguments required by the user-defined AutoMod function. The array must have as many values as the AutoMod function has arguments (any extra values are ignored). The variant values are automatically converted to the type of the associated argument in the AutoMod function in the model. **Note:** The params syntax element is required. If the AutoMod function you are calling has no arguments, you must still define an empty array.

Table H.1: Description of syntax elements for CallFunction method

Return value

A variant that indicates the return value of the user-defined AutoMod function. The variant value is an integer, real, or string, depending on the type of value returned by the AutoMod function; if the function returns a value other than an integer, real, or string (for example, a resource pointer or motor pointer) then the value is converted to a string before it is returned from the CallFunction method. The CallFunction method does not return a value until the AutoMod function has returned a value.

H.2 CloseModel method

Purpose

Use the CloseModel method with the AutoMod runtime object to close a simulation.

Syntax

amx.CloseModel()

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
CloseModel	The name of the method.

Table H.2: Description of syntax elements for CloseModel method

Return value

The CloseModel method does not return a value.

H.3 DisplayView method

Purpose

Use the DisplayView method with the AutoMod runtime object to change the view in the simulation.

Syntax

amx.DisplayView(viewName)

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
DisplayView	The name of the method.
viewName	A string that indicates the name of the AutoMod view that you want to display. (The view must also be defined in the AutoMod model to be displayed during the simulation.)

Table H.3: Description of syntax elements for DisplayView method

Return value

The DisplayView method does not return a value.

H.4 GetVariable method

Purpose

Use the GetVariable method with the AutoMod runtime object to get the current value of an AutoMod variable in the simulation.

Note: If the indicated AutoMod variable does not exist, a message is printed to the Message window and the method throws an exception.

Syntax

```
amx.GetVariable(varName)
```

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
GetVariable	The name of the method.
varName	A string that indicates the name of the AutoMod variable from which you want to obtain a value.

Table H.4: Description of syntax elements for GetVariable method

Note: If you are getting the value of an arrayed variable, there cannot be any spaces in the index of the array element from which you are getting a value. For example, the following GetVariable method is defined *correctly*:

```
Call amx.GetVariable("Vrate(1,3)")
```

The following GetVariable method is defined *incorrectly*:

```
Call amx.GetVariable("Vrate ( 1 , 3 )")
```

Return value

A variant that indicates the value of the AutoMod variable. The variant value is an integer, real, or string, depending on the type of value returned from the AutoMod variable; if the variable is not one of these types (for example, an entity pointer), then the value is converted to a string before it is returned.

H.5 OpenLogFile method

Purpose

Use the OpenLogFile method with the AutoMod runtime object to create a diagnostic file for debugging. The log file contains diagnostic information that is generated during the simulation; it is useful for debugging errors associated with the custom interface.

You can call the OpenLogFile method before the OpenModel method to create a file containing all diagnostic information generated during the run.

If an old log file with the same name already exists, it is replaced when the new file is generated.

Syntax

```
amx.OpenLogFile(fileName)
```

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
OpenLogFile	The name of the method.
fileName	A string that indicates the path and name of the diagnostic file. If you do not specify an absolute path, the location where the file is saved depends on the scripting environment you are using to create the custom interface. See the documentation that came with your scripting environment for more information.

Table H.5: Description of syntax elements for OpenLogFile method

Return value

The OpenLogFile method does not have a return value.

H.6 OpenModel method

Purpose

Use the OpenModel method with the AutoMod runtime object to open a simulation.

Important: The method returns immediately, however, you must wait until the simulation has finished opening before accessing any methods or properties; otherwise, they will fail and throw an exception. The OnModel-Ready event is sent when the simulation has finished opening.

Syntax

amx.OpenModel(modelName, modelPath, cmdLineArgs, ASIEnvVariable, reserved)

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
OpenModel	The name of the method.
modelName	A string that indicates the name of the AutoMod model that you want to run (you do not need to include the .exe extension).
modelPath	A string that indicates the absolute path to the directory containing the AutoMod model that you want to run (the path does not need to include the model name, which is a separate argument). This value cannot be null (you must define the model path).
cmdLineArgs	A string that indicates the command line options you want to use for running the model (for a complete list of the available command line options, see "Using command line options" in the "Running a Model" chapter in volume 1 of the *AutoMod User's Guide*). If this value is null, the simulation runs without any command line options; this is the same as if the model were run from the standard AutoMod interface. **Note:** Command line options must be defined exactly as they are from a command prompt (the options must be preceded by a hyphen).
ASIEnvVariable	A string that indicates an absolute path to the build of the AutoMod software that you want to use for running the model. If this value is null, the software uses the most recently installed build of the same version of the AutoMod software as the type library you have referenced.
reserved	A variant that is reserved for future use; the value must be Nothing.

Table H.6: Description of syntax elements for OpenModel method

Return value

The OpenModel method does not have a return value.

H.7 SetVariable method

Purpose

Use the SetVariable method with the AutoMod runtime object to set the value of an AutoMod variable in the simulation.

Syntax

amx.SetVariable(varName, value)

The syntax elements are defined as follows:

Syntax Element	Description
amx	An object variable that refers to the AutoMod runtime object.
SetVariable	The name of the method.
varName	A string that indicates the name of the AutoMod variable for which you want to set a value. **Note:** If you are setting an arrayed variable, there cannot be any spaces in the index of the array element for which you are setting a value. For example, the following SetVariable method is defined *correctly*: Call amx.SetVariable("Vrate(1,3)", 10) The following SetVariable method is defined *incorrectly*: Call amx.SetVariable("Vrate (1 , 3)", 10)
value	A variant that defines the value to which the variable is set. The variant value must be an integer, real, or string, depending on the type of the AutoMod variable. The AutoMod software attempts to convert the variant value to the correct type for the variable; an exception results if the conversion is not possible or if the variable does not exist.

Table H.7: Description of syntax elements for SetVariable method

Return value

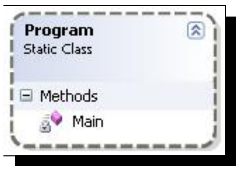
The SetVariable method does not have a return value.

Appendix I

Middleware Classes

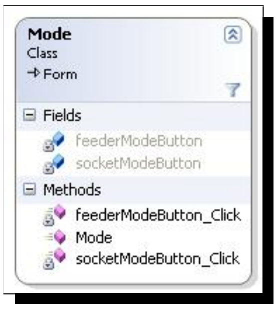
In this appendix we will describe each class in the middleware. We will describe not describe the trivial attributes in the classes (the shaded ones); these attributes are GUI-elements, and some of them have action listeners attached and implemented in methods wich will be described.

I.1 Program



The Program class is the one which starts the middleware application. It contains only the one method: main. This method instantiates an object from the Mode class, wherein the user can select which mode the middleware shall be run in.

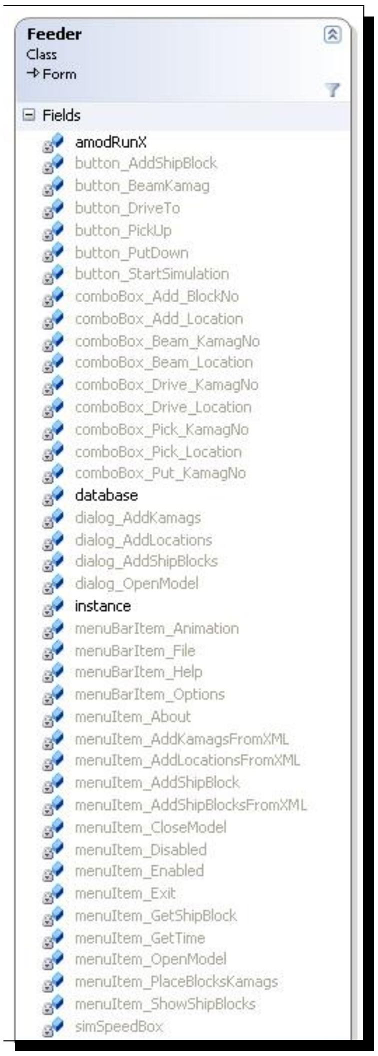
I.2 Mode



The Mode class contains two methods for handling the event, when a user selects one of the two possible modes, namely feeder mode or socket mode. A description of these methods are given next:

- Mode: The constructor of the class, which will start an show the GUI seen in figure 13.2.
- **feederModeButton_Click:** Will hide the Mode-selection-GUI shown in figure 13.2 and start the middleware in feeder mode (see figure 13.4)
- **socketModeButton_Click:** Will hide the Mode-selection-GUI shown in figure 13.2 and start the middleware in socket mode (see figure 13.5)

I.3 Feeder



Feeder
Class
→ Form

Fields
- amodRunX
- button_AddShipBlock
- button_BeamKamag
- button_DriveTo
- button_PickUp
- button_PutDown
- button_StartSimulation
- comboBox_Add_BlockNo
- comboBox_Add_Location
- comboBox_Beam_KamagNo
- comboBox_Beam_Location
- comboBox_Drive_KamagNo
- comboBox_Drive_Location
- comboBox_Pick_KamagNo
- comboBox_Pick_Location
- comboBox_Put_KamagNo
- database
- dialog_AddKamags
- dialog_AddLocations
- dialog_AddShipBlocks
- dialog_OpenModel
- instance
- menuBarItem_Animation
- menuBarItem_File
- menuBarItem_Help
- menuBarItem_Options
- menuItem_About
- menuItem_AddKamagsFromXML
- menuItem_AddLocationsFromXML
- menuItem_AddShipBlock
- menuItem_AddShipBlocksFromXML
- menuItem_CloseModel
- menuItem_Disabled
- menuItem_Enabled
- menuItem_Exit
- menuItem_GetShipBlock
- menuItem_GetTime
- menuItem_OpenModel
- menuItem_PlaceBlocksKamags
- menuItem_ShowShipBlocks
- simSpeedBox

The Feeder class has tree non-GUI attributes which are:

* **amodRunX:** an object instance of the class AmodRunX, which contains the AutoMod ActiveX Object.

* **database:** an object instance of the class Database, which contains data about ship blocks, locations and KAMAGs.

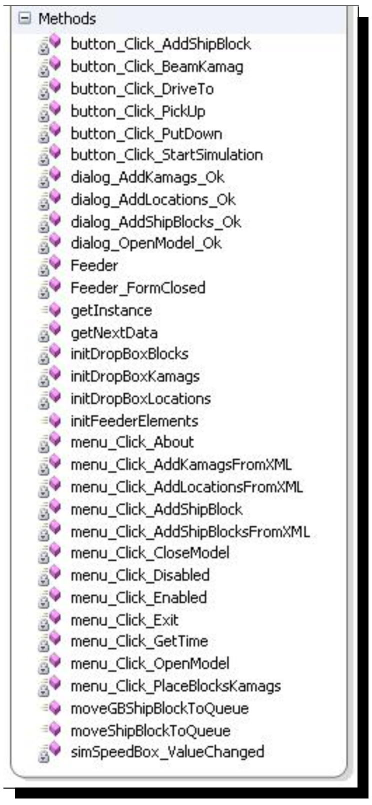
* **instance:** The Feeder class is a singleton class and the instance variable holds this single instance of the Feeder object.

The methods in the Feeder class (shown on next page) does following:

- **button_Click_AddShipBlock:** This method is executed when a user clicks the button "Add Ship Block" (see figure 13.4). It extracts the Block No and Location from the drop down boxes just above the button; and calls the method moveShipBlockToQueue.

- **button_Click_BeamKamag:** This method is executed when a user clicks the button "Beam Kamag" (see figure 13.4). It extracts the KAMAG No and Location from the drop down boxes just above the button; and calls the method beamKamag located in the AmodRunX class.

- **button_Click_DriveTo:** This method is executed when a user clicks the button "Drive To" (see figure 13.4). It extracts the KAMAG No and Location from the drop down boxes just above the button; and calls the method driveToLocation located in the AmodRunX class.



Methods
button_Click_AddShipBlock
button_Click_BeamKamag
button_Click_DriveTo
button_Click_PickUp
button_Click_PutDown
button_Click_StartSimulation
dialog_AddKamags_Ok
dialog_AddLocations_Ok
dialog_AddShipBlocks_Ok
dialog_OpenModel_Ok
Feeder
Feeder_FormClosed
getInstance
getNextData
initDropBoxBlocks
initDropBoxKamags
initDropBoxLocations
initFeederElements
menu_Click_About
menu_Click_AddKamagsFromXML
menu_Click_AddLocationsFromXML
menu_Click_AddShipBlock
menu_Click_AddShipBlocksFromXML
menu_Click_CloseModel
menu_Click_Disabled
menu_Click_Enabled
menu_Click_Exit
menu_Click_GetTime
menu_Click_OpenModel
menu_Click_PlaceBlocksKamags
moveGBShipBlockToQueue
moveShipBlockToQueue
simSpeedBox_ValueChanged

- **button_Click_PickUp:** This method is executed when a user clicks the button "PickUp Ship Block" (see figure 13.4). It extracts the KAMAG No and Location from the drop down boxes just above the button; and calls the method pickUpLoad located in the AmodRunX class.

- **button_Click_PutDown:** This method is executed when a user clicks the button "PickDown Ship Block" (see figure 13.4). It extracts the KAMAG No and Location from the drop down boxes just above the button; and calls the method putDownLoad located in the AmodRunX class.

- **button_Click_StartSimulation:** This method is executed when a user clicks the big button "Start Simulation" (see figure 13.4). It will enable all the other buttons which are disabled at first and set the state of the simulation to not-paused, that is the simulation will start. Another click on the same button will pause the simulation, and yet another click will resume the simulation and so on.

- **dialog_AddKamags_Ok:** This method is executed when the user has selects an XML file containing KAMAGs and details about these whereafter he presses OK in the file dialog. This method will then process the XML file by extracting all the data in it, and add it to the database.

- **dialog_AddLocations_Ok:** This method is executed when the user has selects an XML file containing locations at OSS and details about these whereafter he presses OK in the file dialog. This method will then process the XML file by extracting all the data in it, and add it to the database.

- **dialog_AddShipBlocks_Ok:** This method is executed when the user has selects an XML file containing ship blocks at OSS and details about these whereafter he presses OK in the file dialog. This method will then process the XML file by extracting all the data in it, and add it to the database.

- **dialog_OpenModel_Ok:** This method is executed when the user has selects an exe file which is the AutoMod compiled simulation model, whereafter he presses OK in the file dialog. This method will then instantiate a AmodRunx object (see description in AmodRunX class).
- **Feeder:** This is the constructor of the Feeder class, which initializes the GUI components.
- **Feeder_FormClosed:** Is executed when the user closes the Feeder GUI window with the cross in the upper right corner of the GUI window. This will close the GUI and exit the middleware application.
- **getInstance:** The Feeder class is a singleton and this method is used to get this single object.
- **getNextData:** Used to read data out of XML file.
- **initDropBoxBlocks:** Reads all ship blocks from database and inserts them in a given drop down box in the Feeder GUI.
- **initDropBoxKamags:** Reads all KAMAGs from database and inserts them in a given drop down box in the Feeder GUI.
- **initDropBoxLocations:** Reads all locations from database and inserts them in a given drop down box in the Feeder GUI.
- **initFeederElements:** Uses the three methods above, to insert data to the dropdown boxes in Feeder GUI.
- **menu_Click_About:** Shows information about the current state of the middleware application and version number.
- **menu_Click_AddKamagsFromXML:** This method is executed when the user clicks on the menu item “Add KAMAGs from XML file” in the “Options” field in the menu bar in the Feeder GUI. It will open a file dialog where the user will be able to select and XML file from the file system. This XML file will then be processed by the method dialog_AddKamags_Ok when the user presses OK in the file dialog.
- **menu_Click_AddKamagsFromXML:** This method is executed when the user clicks on the menu item “Add Locations from XML file” in the “Options” field in the menu bar in the Feeder GUI. It will open a file dialog where the user will be able to select and XML file from the file system. This XML file will then be processed by the method dialog_AddLocations_Ok when the user presses OK in the file dialog.
- **menu_Click_AddShipBlock:** This method is executed when the user clicks on the menu item “Add Ship Block” in the “Options” field in

the menu bar in the Feeder GUI. It will open a new GUI window where the user can enter data to a new ship block, which then will be inserted in the database.

- **menu_Click_AddShipBlocksFromXML:** This method is executed when the user clicks on the menu item "Add Ship Blocks from XML file" in the "Options" field in the menu bar in the Feeder GUI. It will open a file dialog where the user will be able to select and XML file from the file system. This XML file will then be processed by the method dialog_AddShipBlocks_Ok when the user presses OK in the file dialog.

- **menu_Click_CloseModel:** Closes the open AutoMod simulation model, when the user clicks on the menu item "Close Model" in the "File" field in the menu bar in the Feeder GUI.

- **menu_Click_Disabled:** Disables the animation in the opened AutoMod simulation model, when the user clicks on the menu item "Disabled" in the "Animation" field in the menu bar in the Feeder GUI.

- **menu_Click_Enabled:** Enables the animation in the opened AutoMod simulation model, when the user clicks on the menu item "Enabled" in the "Animation" field in the menu bar in the Feeder GUI.

- **menu_Click_Exit:** This will close the GUI and exit the middleware application, when the user clicks on the menu item "Exit" in the "File" field in the menu bar in the Feeder GUI.

- **menu_Click_GetTime:** Show the current time in the AutoMod simulation. Is executed when the user clicks on the menu item "Get Simulation Time" in the "Options" field in the menu bar in the Feeder GUI.

- **menu_Click_OpenModel:** Is executed when the user clicks on the menuitem "Open Model" in the "File" field in the menu bar in the Feeder GUI. This method will open a file dialog where the user can select an AutoMod compiled simulation model file with exe extension.

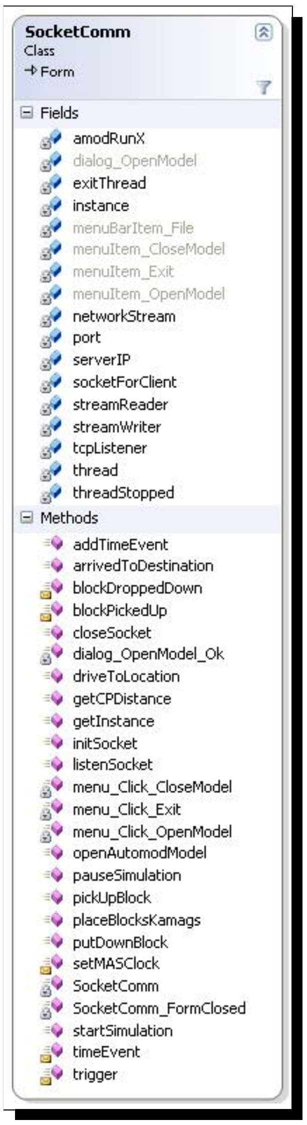
- **menu_Click_PlaceBlocksKamags:** Used to initialize the simulation model, by inserting ship blocks and KAMAGs at appropriate locations before simulation start.

- **moveGBShipBlockToQueue:** Used to insert a ship *Grand* block to a location in the simulation.

- **moveShipBlockToQueue:** Used to insert a ship block to a location in the simulation.

- **simSpeedBox_ValueChanged:** Is executed whenever the speed of the simulation is changed in the Animation Display Step Box. This will increase or decrease the simulation speed in the AutoMod simulation model.

I.4 SocketComm



SocketComm
Class
→ Form

Fields
amodRunX
dialog_OpenModel
exitThread
instance
menuBarItem_File
menuItem_CloseModel
menuItem_Exit
menuItem_OpenModel
networkStream
port
serverIP
socketForClient
streamReader
streamWriter
tcpListener
thread
threadStopped

Methods
addTimeEvent
arrivedToDestination
blockDroppedDown
blockPickedUp
closeSocket
dialog_OpenModel_Ok
driveToLocation
getCPDistance
getInstance
initSocket
listenSocket
menu_Click_CloseModel
menu_Click_Exit
menu_Click_OpenModel
openAutomodModel
pauseSimulation
pickUpBlock
placeBlocksKamags
putDownBlock
setMASClock
SocketComm
SocketComm_FormClosed
startSimulation
timeEvent
trigger

The SocketComm class has tree non-GUI attributes which are:

* **amodRunX:** an object instance of the class AmodRunX, which contains the AutoMod ActiveX Object.

* **exitThread:** A boolean used to exit the thread which listens to incoming socket connections.

* **instance:** The SocketComm class is a singleton class and the instance variable holds this single instance of the SocketComm object.

* **networkStream:** Used by the streamReader and streamWriter to receive or send socket messages.

* **port:** Defines the socket comunication port.

* **serverIP:** Defines the socket comunication IP address, which will be "127.0.0.1" if middleware application and the MAS runs on localy.

* **socketForClient:** Used to check whenever af MAS connects to the middleware socket server and to initialize the networkStream.

* **streamReader:** Used to receive a socket message.

* **streamWriter:** Used to send a socket message.

* **tcpListener:** The socket server itselft, that is the middleware TCP/IP socket server which accepts connections from the MAS.

* **thread:** A thread that continually listens for socket connections.

* **threadStopped:** A boolean indicating whether the thread is stopped, that is whether the TCP/IP socket server has stopped listening. This is necessary in exitting the middleware application, because the socket server has to be stopped before the middleware application can be exited, or else there will occur an exception.

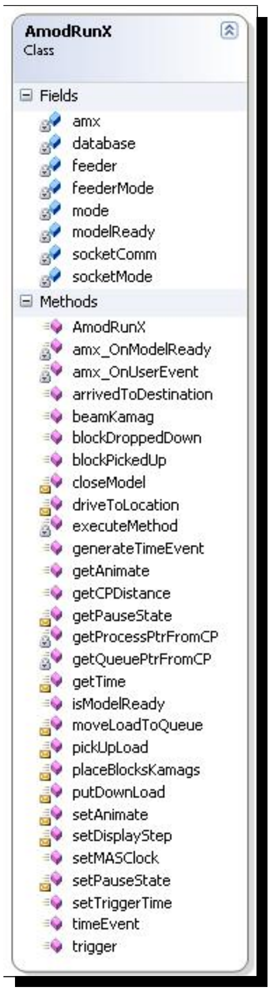
The methods in the SocketComm class (shown on previos page) and their descriptions are listed below. But first we will mention that methods with blue color are executed from the MAS, by the help of sockets and reflection. The methods with green color indicate methods that are executed from the simulation by AutoMod's activeX runtime object and reflection.

- **addTimeEvent:** Executes the addTimeEvent defined in the AmodRunX class. See this method for further details.
- **arrivedToDestination:** This method informs the MAS, that a specific KAMAG has arrived to specific location in the simulation model. This information is sent by the a socket message.
- **blockDroppedDown:** This method informs the MAS, that a specific KAMAG has dropped a ship block to specific location in the simulation model. This information is sent by the a socket message.
- **blockPickedUp:** This method informs the MAS, that a specific KAMAG has picked up a ship block from a specific location in the simulation model. This information is sent by the a socket message.
- **closeSocket:** Closes all streams and connections and finally the socket server.
- **dialog_OpenModel_Ok:** This method is executed when the user has selects an exe file which is the AutoMod compiled simulation model, whereafter he presses OK in the file dialog. This method will then instantiate a AmodRunx object (see description in AmodRunX class).
- **driveToLocation:** Executes the driveToLocation defined in the AmodRunX class. See this method for further details.
- **getCPDistance:** Is used by the MAS to calculate the distance between two "Control Points", that is between two locations at OSS.
- **getInstance:** The SocketComm class is a singleton and this method is used to get this single object.
- **initSocket:** Sets the IP and Port number of the TCP/IP socket server and starts the server for listening to incoming connections.

- **listenSocket:** Listens for incoming connections from the MAS. When MAS connects, then this method receives a socket message, which will be parsed to a method call with a method name and parameters, which will be invoked with the help of reflection if the method exists.
- **menu_Click_CloseModel:** Closes the open AutoMod simulation model, when the user clicks on the menu item "Close Model" in the "File" field in the menu bar in the SocketComm GUI.
- **menu_Click_Exit:** This will close the GUI and exit the middleware application, when the user clicks on the menu item "Exit" in the "File" field in the menu bar in the SocketComm GUI.
- **menu_Click_OpenModel:** Is executed when the user clicks on the menuitem "Open Model" in the "File" field in the menu bar in the SocketComm GUI. This method will execute the method openAutomodModel beneath and start the TCP/IP socket server.
- **openAutomodModel:** Is executed from the menu_Click_OpenModel method. This method will open a file dialog where the user can select an AutoMod compiled simulation model file with exe extension.
- **pauseSimulation:** Pauses the simulation with the setPauseState method located in the AmodRunX class.
- **pickUpBlock:** Executes the pickUpBlock defined in the AmodRunX class. See this method for further details.
- **placeBlocksKamags:** Executes the placeBlocksKamags defined in the AmodRunX class. See this method for further details.
- **putDownBlock:** Executes the putDownBlock defined in the AmodRunX class. See this method for further details.
- **setMASClock:** Used to synchronize MAS clock with AutmoMod Simulation clock.
- **SocketComm:** The constructor of the SocketComm class. Initializes the SocketComm GUI, the boolean attributes and the TCP/IP socket server.
- **SocketComm_FormClosed:** Is executed when the user closes the SocketComm GUI window with the cross in the upper right corner of the GUI window. This will close the GUI and exit the middleware application.
- **startSimulation:** Starts the simulation with the setPauseState method located in the AmodRunX class.

- **timeEvent:** Notifies the MAS, that a specific time has been reached. This timeevent was added to be generated with the help of the method addTimeEvent.
- **trigger:** Triggers the MAS, whithin some prior specified time intervals, so that the D-Planner can coordinate and distribute transportation tasks at these trigger intervals.

I.5 AmodRunX



AmodRunX
Class

Fields
amx
database
feeder
feederMode
mode
modelReady
socketComm
socketMode

Methods
AmodRunX
amx_OnModelReady
amx_OnUserEvent
arrivedToDestination
beamKamag
blockDroppedDown
blockPickedUp
closeModel
driveToLocation
executeMethod
generateTimeEvent
getAnimate
getCPDistance
getPauseState
getProcessPtrFromCP
getQueuePtrFromCP
getTime
isModelReady
moveLoadToQueue
pickUpLoad
placeBlocksKamags
putDownLoad
setAnimate
setDisplayStep
setMASClock
setPauseState
setTriggerTime
timeEvent
trigger

The AmodRunX class has following attributes:

* **amx:** Is the AutoMod Runtime Object, which is used to execute AutoMod simulation functions, receive AutoMod simulation events and to set/get variables from the AutoMod simulation, with the use of the AutoMod ActiveX component.
* **database:** an object instance of the class Database, which contains data about ship blocks, locations and KAMAGs.
* **feeder:** If middleware is run in Feeder mode, then this attribute will reference to the feeder object.
* **feederMode:** A constant string, used to check which mode the middleware is in.
* **mode:** A string which will be set to either the attribute feederMode or socketMode depending on which mode the middleware is run in.
* **modelReady:** A boolean which represents when the simulation model read and ready.
* **socketComm:** If middleware is run in Socket mode, then this attribute will reference to the socketComm object.
* **socketMode:** A constant string, used to check which mode the middleware is in.

The AmodRunX methods are:

- **addTimeEvent:** Used to add a specific time into the simulation, so that the simulation will generate a timeevent at this specified time.

- **AmodRunX:** The constructor of the class, which
    1. sets the mode to either feeder mode or socket mode
    2. instantiates the database object
    3. instantiates the amx object
    4. add event listeners to the amx object
    5. opens the simulation model with the amx object.
- **amx_OnModelReady:** Executed when the simulation model is opened and ready. This method will enable the animation of the simulation, and set the modelready attribute to true, to indicate that the simulation is ready to be run.
- **amx_OnUserEvent:** Receives events from the simulation model, in the form of a comma separated string, and executes the method executeMethod (see this method for further description).
- **arrivedToDestination:** If middleware is run in socket mode, then this method will inform the MAS, whenever a KAMAG arrives to a location in the simulation model.
- **beamKamag:** Beams a KAMAG in the simulation model from location to another location. Used when the simulation model is initialized to place KAMAGs.
- **blockDroppedDown:** If middleware is run in socket mode, then this method will inform the MAS, whenever a KAMAG has dropped a ship block to a location in the simulation model.
- **blockPickedUp:** If middleware is run in socket mode, then this method will inform the MAS, whenever a KAMAG has picked up a ship block from a location in the simulation model.
- **closeModel:** sets the modelReady attribute to false and closes the amx object, that is closes simulation model.
- **driveToLocation:** Calls an AutoMod function in the simulation model, which informs a KAMAG to drive to a location in the simulation model.
- **executeMethod:** Receives af comma separated string from the method amx_OnUserEvent which will be parsed to a method with parameters, and invoked by the help of reflection.
- **getAnimate:** Returns an integer value indication whether the animation in the simulation model is enabled or not.

- **getCPDistance:** Given two locations in the simulation model, this method will return the distance between them, by the use of a function from the AutoMod simulation model.

- **getPauseState:** Returns a boolean value indicating whether the simulation is running or not.

- **getProcessPtrFromCP:** A method which is used from the simulation model to get the ProcessPtr variable, which belongs to the given Control Point (location).

- **getQueuePtrFromCP:** A method which is used from the simulation model to get the QueuePtr variable, which belongs to the given Control Point (location).

- **getTime:** Returns the simulation model time (absolute clock, which as default is 0 at simulation start).

- **isModelReady:** Return a true/false indicating whether the simulation model is ready or not.

- **moveShipBlockToQueue:** This method adds a ship block to a location in the simulation model. To use this method the following arguments has to be given:

    1. Location
    2. Block No
    3. Block Weight
    4. Block Length
    5. Block Breadth
    6. Block Height
    7. Grand Block No
    8. Block Family

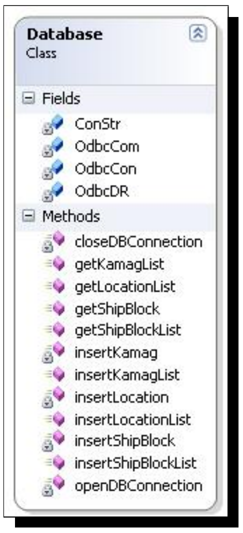
    The arguments 3-8 can be found in the database, if the ship block with identification number "argument 2" exists in the database.

- **pickUpBlock:** Calls an AutoMod function in the simulation model, which informs a KAMAG to pick up a ship block from the KAMAGs current location in the simulation model.

- **placeBlocksKamags:** Used to initialize the simulation model. Given two comma separated strings; one of ship blocks/locations and another of KAMAGs/locations, this method places the ship blocks and KAMAGs at the specified locations in the simulation model.

- **putDownBlock:** Calls an AutoMod function in the simulation model, which informs a KAMAG to put down a ship block to the KAMAGs current location in the simulation model.
- **setAnimate:** Enables/Disables the animation in the simulation.
- **setDisplayStep:** Sets the displaystep (Simulation speed), with a numeric value.
- **setMASClock:** If middleware is run in socket mode, then this method will inform the MAS about the clock in the simulation at periodic intervals, so that the MAS clock is synchronized with the simulation clock.
- **setPauseState:** Pauses/resumes the simulation.
- **setTriggerTime:** Sets a time in the simulation by the use of a AutoMod function, so that a timeevent is generated at that time.
- **timeEvent:** If middleware is run in socket mode, then this method will inform the MAS that a specific time has been reached. This time was formerly set by the MAS, which wanted to know when this time was reached.
- **trigger:** If middleware is run in socket mode, then this method will trigger the MAS, so that the planning agent (D-planner) can calculate/coordinate transportation requests from C-planners.

I.6 Database



The Database class has following attributes:

∗ **ConStr:** Is the Connection String, which contains following information:

1. Which driver to use to connect to the databse, in our case a MySQL ODBC driver.
2. The database server IP, in our case "localhost" (could also be a remote database server)
3. The database port, in our case 3306
4. The database schema, in our case oss
5. The database username, in our case oss
6. The database password, in our case sso

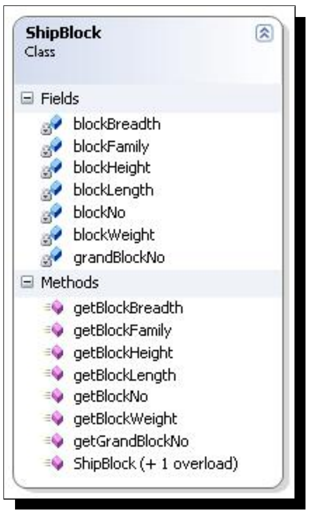
7. An a database option, which is set to 3 (a default value).

* **OdbcCom:** Used to execute SQL commands.
* **OdbcCon:** Uses the OdbcStr to connect to the database, and is used by the OdbcCom to connect to the database.
* **OdbcDR:** When OdbcCom executes a SQL query, then the result is stored in this attribute.

The Database method are:

- **closeDBConnection:** Closes the database connection
- **getKamagList:** Returns all the KAMAGs at OSS from the databse in a list.
- **getLocationList:** Returns all the locations at OSS from the databse in a list.
- **getShipBlock:** Returns a ship block at OSS from the databse given af ship block No.
- **getShipBlockList:** Returns all the ship blocks at OSS from the databse in a list.
- **insertKamag:** Adds a KAMAG to the database.
- **insertKamagList:** Adds a list of KAMAGs to the database.
- **insertLocation:** Adds a location to the database.
- **insertLocationList:** Adds a list of locations to the database.
- **insertShipBlock:** Adds a ship block to the database.
- **insertShipBlockList:** Adds a list of ship blocks to the database.
- **openDBConnection:** Opens a database connection.

I.7 ShipBlock



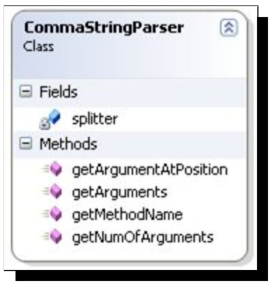
The ShipBlock class has following attributes:

* **blockBreadth:** The breadth of the ship block.
* **blockFamily:** The family of the ship block.
* **blockHeight:** The height of the ship block.
* **blockLength:** The length of the ship block.
* **blockNo:** The block No of the ship block.
* **blockWeight:** The weight of the ship block.
* **grandBlockNo:** The grand block No to which this ship block belongs.

The ShipBlock methods are:

- **getBlockBreadth:** Returns the breadth of the ship block.
- **getBlockFamily:** Returns the family of the ship block.
- **getBlockHeight:** Returns the height of the ship block.
- **getBlockLength:** Returns the length of the ship block.
- **getBlockNo:** Returns the block No of the ship block.
- **getBlockWeight:** Returns the weight of the ship block.
- **getBlockGrandBlockNo:** Returns the grand block No to which this ship block belongs.
- **ShipBlock:** The exists two constructors of the class. One for a ship block and another for a grand ship block, which is a collection of ship blocks.

I.8 CommaStringParser



The CommaStringParser class has following attributes:

* **splitter:** Is a comma character, to split comma strings.

The CommaStringParser methods are:

- **getArgumentAtPosition:** Returns an argument in the comma string, the position has to bigger than 1 and less than the actual arguments.
- **getArguments:** Returns all the arguments in the comma string.
- **getMethodName:** Returns the method name in the comma string, which is the first element in the comma string.
- **getNumOfArguments:** Returns the number of arguments in the comma string.

Appendix J

Journal

Date	Duration	Person	Responsibility	Subject of Meeting
27/01-2006	5 hours	Niels J. Jacobsen	External Supervisor	Introduction and tour at Lindø
27/01-2006	1 hour	Henning K. Jensen	B-planner	Intro to B-plan
27/01-2006	30 min.	Claus Rønaa	D-plan	Intro to D-plan (very short)
07/02-2006	2 hours	Henning K. Jensen	B-planner / Data provider	Insight in B-plan and logistics at Lindø
13/02-2006	3 hours	Niels J. Jacobsen	Extern supervisor	Getting Computer and passwords
21/02-2006	2 hours	Henning K. Jensen	B-planner / Data Provider	Lecture on how to use DPS and Blue print
07/03-2006	3 hours	Ole T. Sørensen	C-planner	His Daily rutine
07/03-2006	1 hour	Charlotte Mølgaard	C-planner	Her Daily rutine
07/03-2006	30 min.	Henning K. Jensen	B-planner / Data provider	Status Project Orientation
24/04-2006	4 hours	Claus Rønaa	D-planner	Insight in D-plan and work process
02/05-2006	2 hours	Claus Rønaa	D-planner	Insight in D-plan and work process
02/05-2006	30 min.	Kaare Black	Lindø Byg	Map of Lindø

Date	Duration	Person	Responsibility	Subject of Meeting
02/05-2006	30 min.	Niels J. Jacobsen	Extern supervisor	Status meeting about thesis
16/05-2006	5 hours	Ivan S. Jensen	Production Engineer at Simcon	Automod and Sockets
16/05-2006		Kasper hallenborg	Supervisor	Status report turn in
16/05-2006		Niels J. Jacobsen	External supervisor	Status report turn in
29/05-2006	1 hour	Henning K. Jensen	Production Engineer at OSS	Provided updated dxf map
31/05-2006	6 hours	Ali C & Henrik MM	Presentation of OSS case	Decide Seminar
03/08-2006 -17/08-2006	100 hours	Ali C & Henrik MM	Learning about MAS	MAS Course
29/08-2006	3 hours	Simcon	Chief Developer at Simcon	System emulation workshop
23/08-2006	1 hours	Yves Demazeau	Coordinater of MAGMA research group	MAS in case
05/09-2006	6 hours	DECIDE members	Decide members	Decide Seminar at OSS
08/11-2006	30 min	Kasper Hallenborg	Supervisor	Presentation discussion
08/11-2006				Turning in Automod dongle
15/11-2006	5 hours			Case Presentation
24/11-2006	1 hour			Case Presentation
27/11-2006	1 hour	Yves Demazeau	Coordinater og MAGMA research group	MAS discussion

Appendix K

Source code

In this appendix we will show code fragments that are essential to the project.

K.1 Reading control point and neighbours from XML file

This code shows how the control point neighbours are read from an XML file.

```
Hashtable controlPoints = new Hashtable();
ArrayList neighbours = new ArrayList();

// Create a resolver with default credentials.
XmlUrlResolver resolver = new XmlUrlResolver();
resolver.Credentials = System.Net.CredentialCache.
    DefaultCredentials;

XmlReaderSettings settings = new XmlReaderSettings();
// Set the reader settings object to use the resolver.
settings.XmlResolver = resolver;

String filePath = "file:///" + dialog_AddDistances.FileName;
//MessageBox.Show(filePath);
//FileInfo fileInfo = new FileInfo(filePath);

// Create the XmlReader object.
XmlReader globalReader = XmlReader.Create(filePath, settings
    );

// read the text content of the elements.
globalReader.Read();
globalReader.ReadToNextSibling("Workbook");
globalReader.ReadToDescendant("Worksheet");
globalReader.ReadToDescendant("Table");
globalReader.ReadToDescendant("Row");
```

```
      String controlPoint;
      String neighbour;

      //Start to process each control point
      while (globalReader.Name.Equals("Row"))
      {
        neighbours = new ArrayList();
        XmlReader reader = globalReader.ReadSubtree();
        reader.ReadToDescendant("Cell");
        reader.ReadToDescendant("Data");
        controlPoint = reader.ReadString();
        reader.Skip();

        while (reader.ReadToNextSibling("Cell"))
        {
          reader.ReadToDescendant("Data");
          neighbour = reader.ReadString();
          reader.Skip();

          neighbours.Add(neighbour);
        }

        controlPoints.Add(controlPoint, neighbours);

        reader.Close();
        globalReader.ReadToNextSibling("Row");
      }

      database.insertDistances(controlPoints);
    }
```

K.2 Getting and inserting control point distances

The method shown below will iterate through all the keys (control points) in the given hashmap, and for the corresponding value (a list of neighbour control points), find the distance to each neighbour by calling the method “getCPDistance”, which will call an AutoMod function that will return the distance between two control points. The control point, its neighbour an their distance are then added to the MySQL database with the method “insertDistance”.

```
internal void insertDistances(Hashtable controlPoints)
{
  openDBConnection();

  String sqlCommand = "CREATE TABLE IF NOT EXISTS distance" +
      " (" +
      " CP1 VARCHAR(255) NOT NULL," +
      " CP2 VARCHAR(255) NOT NULL," +
      " distance DOUBLE NOT NULL," +
      " PRIMARY KEY (CP1,CP2)" +
      ")";
  OdbcCom = new OdbcCommand(sqlCommand, OdbcCon);
  OdbcCom.ExecuteNonQuery();

  foreach (DictionaryEntry d in controlPoints)
  {
    String controlPoint = (String)d.Key;
    ArrayList neighbours = (ArrayList)d.Value;
    String distance;

    foreach (String neighbour in neighbours)
    {
      distance = Feeder.getInstance().getCPDistance(controlPoint,
          neighbour);
      insertDistance("CP_"+controlPoint, "CP_"+neighbour, distance
          );
    }
  }
  closeDBConnection();
}

private void insertDistance(String controlPoint, String neighbour,
    String distance)
{
  String sqlCommand = "REPLACE INTO distance" +
      " VALUES ('" + controlPoint + "'," + "'" + neighbour + "',"
          + distance + ")";
  OdbcCom = new OdbcCommand(sqlCommand, OdbcCon);
  OdbcCom.ExecuteNonQuery();
}
```

K.3 Coordinate of control point on arc path

This code shows how the coordinate of a control point located on an arc path is calculated.

```
  //Translate (cenX,ceny) to (0,0) and (begX,begY) accordingly
  if (cenX > 0)
    begX0 = begX − cenX;
  else
    begX0 = begX + cenX;
  if (centerY > 0)
    begY0 = begY − cenY;
  else
    begY0 = begY + cenY;

  //Calculate radius og circumference
  radius = Math.Sqrt(Math.Pow(begX0, 2) + Math.Pow(begY0, 2));
  circumference = 2 * Math.PI * radius;
```

```
//Translate (begX0,begY0) to the unit circle
begX0Unit = begX0 / radius;
begY0Unit = begY0 / radius;

//Calculate begAngle
begAngle = Math.Acos(begX0Unit) * (180 / Math.PI);

//Calculate begCPangle, it will only be the absolute value
begCPangle = distance * (360 / circumference);

//Calculate CPangle
if (angle < 0)
  CPangle = begAngle - begCPangle;
else
  CPangle = begAngle + begCPangle;

//Calculate the coordinate of the control point
//in the unit circle
coordX0Unit = Math.Cos(CPangle * (Math.PI / 180));
coordY0Unit = Math.Sin(CPangle * (Math.PI / 180));

//Multiply unit coordinates with radius to translate back
coordX0 = coordX0Unit * radius;
coordY0 = coordY0Unit * radius;

//Translate (cenX,cenY) back to original coordinates, and
//translate (coordX0,coordY0) accordingly to get (coordX,coordY)
if (centerX > 0)
  coordX = coordX0 + cenX;
else
  coordX = coordX0 - cenX;
if (centerY > 0)
  coordY = coordY0 + cenY;
else
  coordY = coordY0 - cenY;
```

K.4 Coordinate of control point on straight line path

This code shows how the coordinate of a control point located on a straight line path is calculated.

```
    //Check whether it is a vertical line
    if (begX == endX)
    {
      coordX = begX;
      if (endY > begY)
        coordY = begY + d;
      else
        coordY = begY - d;
    }
    //The line has a slope, it's not a vertical line
    else
```

```
        {
          //find lineSlope (a) and yIntercept (b) in equation y=ax+b
          lineSlope = (endY − begY) / (endX − beginX);
          yIntercept = begY − lineSlope * beginX;

          //Solve second degree equation
          A = 1 + Math.Pow(lineSlope , 2);
          B = 2 * lineSlope * (yIntercept − begY) − 2 * begX;
          C = Math.Pow(begX, 2) + Math.Pow((yIntercept − begY), 2) −
               Math.Pow(distance , 2);

          coordX1 = (−B + Math.Sqrt(Math.Pow(B, 2) − 4 * A * C)) /
              (2 * A);
          coordX2 = (−B − Math.Sqrt(Math.Pow(B, 2) − 4 * A * C)) /
              (2 * A);

          //Pick the solution for coordX that lies between begX og
              endX
          if (endX > begX)
          {
            if (coordX1 >= begX && coordX1 <= endX)
              coordX = coordX1;
            else if (coordX2 >= begX && coordX2 <= endX)
              coordX = coordX2;
          }
          else
          {
            if (coordX1 <= begX && coordX1 >= endX)
              coordX = coordX1;
            else if (coordX2 <= begX && coordX2 >= endX)
              coordX = coordX2;
          }

   coordY = lineSlope * coordX + yIntercept;
```

K.5 Middleware socket communication and reflection

The code below shows how the middleware handles method calls from the Multi-Agent System.

```
public void listenSocket()
{
  try
  {
    //Check if there is a client (MAS), which wants to connect
    while (!tcpListener.Pending() && !exitThread)
    {
      Thread.Sleep(10);
    }
    if (!exitThread)
    {
      socketForClient = tcpListener.AcceptSocket();
      networkStream = new NetworkStream(socketForClient);
      streamWriter = new StreamWriter(networkStream);
      streamReader = new StreamReader(networkStream);

      if (socketForClient.Connected)
      {
        String theString = "";
        try
        {
          theString = streamReader.ReadLine();

    //Extract methodname and arguments to method
          String methodName = CommaStringParser.getMethodName(
              theString);
          String[] arguments = CommaStringParser.getArguments(
              theString);

          Type objectType = this.GetType();
          MethodInfo methodInfo = objectType.GetMethod(methodName)
              ;

          if (methodInfo == null)
          {
            MessageBox.Show("Ingen Metoder der hedder: " +
                theString);
            streamWriter.WriteLine("unknown method");
            streamWriter.Flush();
          }
          else
          {
      //Execute method
            methodInfo.Invoke(this, arguments);
          }
    //Wait for another method call
          listenSocket();
        }
        catch (System.IO.IOException)
        {
          MessageBox.Show("Client (MAS) has quit, Middleware will
              exit now");
          closeSocket();
          Application.Exit();
        }
      }
    }
  }
  finally
  {
    threadStopped = true;
  }
}
```

K.6 Middleware AutoMod communication and reflection

The code below shows how the middleware handles method calls from the running AutoMod model.

```
private void amx_OnUserEvent(int i, String str)
{
  switch (i)
  {
    case 0://A method call contained in the string str that is
        comma separated
      executeMethod(str);
      break;
    case 1:
      amx.SetVariable("V_animating", getAnimate());
      break;
    case 2:
      // A request to convert a Location to a QueuePtr
      getQueuePtrFromCP();
      break;
    case 3:
      // Her får vi en request på at få konverteret en Location om
          til en ProcessPtr
      getProcessPtrFromCP();
      break;
  }
}

private void executeMethod(String str)
{
  String methodName = CommaStringParser.getMethodName(str);
  String[] arguments = CommaStringParser.getArguments(str);

  Type objectType = this.GetType();
  MethodInfo methodInfo = objectType.GetMethod(methodName);

  if (methodInfo == null)
  {
    System.Windows.Forms.MessageBox.Show("Ingen Metoder der hedder
        : " + methodName);
  }
  else
  {
    methodInfo.Invoke(this, arguments);
  }
}
```

K.7 Comma string parser

The Multi-Agent System and the AutoMod model can execute methods in the middleware, which is done by reflection. The middleware receives a comma separated string which it has to parse to execute a method with the parameters contained in the received string. The comma string parser has methods for returning method name and arguments from such a comma string. The code is shown below.

```
class CommaStringParser
{
  private static char[] splitter = { ',' };

  public static int getNumOfArguments(String str)
  {
    int numOfArguments = 0;
    String[] splittetString = str.Split(splitter);

    numOfArguments = splittetString.Length - 1;
    return numOfArguments;
  }

  public static String getArgumentAtPosition(String str, int pos)
  {
    String[] splittetString = str.Split(splitter);
    if (pos > (splittetString.Length - 1))
      return null;
    return splittetString[pos];
  }

  public static String[] getArguments(String str)
  {
    String[] splittetString = str.Split(splitter);
    String[] arguments = new String[splittetString.Length-1];
    for (int i = 1; i < splittetString.Length; i++)
      arguments[i - 1] = splittetString[i];
    return arguments;
  }

  public static string getMethodName(String str)
  {
    String[] splittetString = str.Split(splitter);
    return splittetString[0];
  }
}
```

Bibliography

[1] The foundation for intelligent physical agents. http://www.fipa.org.

[2] Multi-agent systems lab. the distributed vehicle monitoring testbed. http://dis.cs.umass.edu/research/dvmt/.

[3] *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[4] Ansøgning om it-korridor projektet - decide, April 2005.

[5] Bang & olufsen design philosophy, October 2006.

[6] Sibel Adalý and Leo Pigaty. The darpa advanced logistics project. *Annals of Mathematics and Artificial Intelligence*, 37(4):409–452, November 2003.

[7] Zafeer Alibhai. What is Contract Net interaction Protocol?, July 2003.

[8] Eduardo Alonso. In3016/inm326 software agents. multi-agent systems: Communication, 2005.

[9] Sara Baase and Allen Van Gelder. *Computer Algorithms - Introduction to Design and Analysis*. Addison Wesley Longman, 2000.

[10] Fabio Bellifimine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade - A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.

[11] Rafael H. Bordini and Jomi F. Hübner. *A Java-based interpreter for an extended version of AgentSpeak*, February 2007.

[12] David Brackeen, Bret Barker, and Laurence Vanhelswue. *Developing Games in Java*. New Riders Games, 2003.

[13] Norman Carver, Victor Lesser, and Qiegang Long. Distributed Sensor Interpretation: Modeling Agent Interpretations in DRESUN. In *UMass Technical Report, UMCS 93-75*, sep 1993.

[14] Renque Corporation. *Renque User Guide*, 2007.

[15] Michael Pěchouček "David Šišlák, Martin Rehák and Dušan Pavlíček". A-globe: Agent development platform with inaccessibility and mobility support.

[16] Keith Decker. Environment centered analysis and design of coordination mechanisms. Technical Report UM-CS-1995-069, May 1995.

[17] Keith Decker and Jinjiang Li. Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3(2):133–157, 2000.

[18] Yves Demazeau. Usd mip course am/amp24 slides, August 2006.

[19] Computing Laboratory Department of Computing Science. *JavaSim User Guide*, 1999.

[20] A BBN Technologies Document. Cougaar Architecture Document, December 2004.

[21] T. Finin and J. Weber. Draft. specification of the kqml agentcommunication language, 2003.

[22] FIPA. *FIPA Communicative Act Library Specification*. FIPA, 2002.

[23] FIPA. *FIPA Contract Net Interaction Protocol Specification*. FIPA, 2002.

[24] Tiziana Trucco (TILAB formerly CSELT) Giovanni Rimassa (University of Parma) Fabio Bellifemine Giovanni Caire. *JADE PROGRAMMER'S GUIDE*, August 2006.

[25] John Graham, Michael Mersic, and Keith Decker. Scalability and scheduling in an agent architecture.

[26] John R. Graham, Keith S. Decker, and Michael Mersic. Decaf - a flexible multi agent system architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):7–27, 2003.

[27] John Robert Graham. *Real-time scheduling in distributed multi agent systems*. PhD thesis, 2001. Professor In Charge-Keith S. Decker.

[28] Dr. Mark Greaves. Survivable Logistics Information Systems, September 2002.

[29] Kasper Hallenborg and Yves Demazeau. Dynamical control in large-scale material handling systems through agent technology. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 637–645, Washington, DC, USA, 2006. IEEE Computer Society.

[30] Aaron Helsinger, Karl Kleinmann, and Marshall Brinn. A framework to control emergent survivability of multi agent systems. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 28–35, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Bryan Horling, Victor Lesser, Regis Vincent, Tom Wagner, Anita Raja, Shelley Zhang, Keith Decker, and Alan Garvey. The TAEMS White Paper, jan 1999.

[32] INPG IGN. Generalisation modelling using an agent paradigm. Technical Report ESPRIT / LTR / 24 939, 1998.

[33] Acronymics. Inc. An integrated toolkit for constructing intelligent software agents, 2004.

[34] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[35] Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages, 1999.

[36] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Baltimore, MD 21250, 1997.

[37] Jaron Collis Divine Ndumu Intelligent Systems Research Group BT Labs. *ZEUS Technical Manual*, September 1999.

[38] J. C. Collis D. T. Ndumu H. S. Nwana L. C. Lee. The ZEUS Agent Building Tool-kit. *BT Technology Journal*, 16(3):60–68, 1998.

[39] GoldSim Technology Group LLC. *GoldSim User Guide, Probalistic Simulation Environment*, 2007.

[40] Agent Oriented Software Pty. Ltd. Jack intelligent agents - summary of an agent infrastructure.

[41] Agent Oriented Software Pty. Ltd. *JACK Intelligent Agents™, Agent Manual*, 2006.

[42] Foster McGeary. Decaf programming: An introduction. April 2001.

[43] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[44] inc Realtime technologies. *SimCreator User Guide*, 2004.

[45] Brooks Software. *AutoMod 12.0. User's Guide*, 2005.

[46] R. Systems. An integrated toolkit for constructing intelligent software agents, 1999.

[47] Giovanni Caire (TILAB formerly CSELT). *JADE TUTORIAL, JADE PROGRAMMING FOR BEGINNERS*, December 2003.

[48] D. Šišlák, M. Rollo, and M. Pěchouček. A-globe: Agent platform with inaccessibility and mobility support. In M. Klusch, S. Ossowski, V. Kashyap, and R. Unland, editors, *Cooperative Information Agents VIII*, number 3191 in LNAI. Springer-Verlag, Heidelberg, sep 2004.

[49] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In *Artificial Intelligence Planning Systems*, pages 154–163, 1998.

[50] David E. Wilkins, Karen L. Myers, Marie desJardins, and et al. Multiagent planning architecture - mpa version 1.8.

[51] Michael Wooldridge. Practical reasoning with procedural knowledge. In *Formal and Applied Practical Reasoning*, pages 663–678, 1996.

[52] Michael Wooldridge. *Introduction to MultiAgent Systems*. John Wiley and Sons, 2002.