Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's Thesis

Agent-based Task and Resource Allocation Tool

Ondřej Jelínek

Supervisor: Ing. Jiří Vokřínek, Ph.D.

Study Programme: Open Informatics (Master)

Field of Study: Artificial Intelligence

May 12, 2014

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Ondřej Jelínek**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Agent-based Task and Resource Allocation Tool**

Guidelines:

1. Study the agent-based task and resource allocation methods.
2. Implement selected allocation heuristics and algorithms using A-lite toolkit with focus to scheduling and transportation problems.
3. Design and implement graphical tool for agent-based allocation scenarios configuration.
4. Create illustrative examples of allocation scenarios and test implemented tools.

Bibliography/Sources:

Peter Brucker: Scheduling algorithms (4. ed.). Springer 2004, ISBN 978-3-540-20524-1, pp. I-XII, 1-367

Vokrinek, J. and Komenda , A. and Pechoucek , M.: Abstract Architecture for Task-oriented Multi-agent Problem Solving. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 2011, vol. 41, p. 31 -40. ISSN 1094-6977.

Diploma Thesis Supervisor: Ing. Jiří Vokřínek, Ph.D.

Valid until the end of the summer semester of academic year 2014/2015

[illegible]

doc. Ing. Filip Železný, Ph.D.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-
L.S.

[illegible]

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 3, 2014

Aknowledgements

In this place, I would like to thank my supervisor *Ing. Jiří Vokřínek, Ph.D.* for his patient guidance and valuable advices.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

In Prague on May 12, 2014 ....

Abstract

Distributed planning and scheduling plays a major role in many industrial applications ranging from multirobotics systems, military tactical operations to manufacturing and logistics. In this master thesis, we present and describe a multi-agent solver which uses techniques of task sharing, delegation and allocation on locally planned resources. The agent-based task and resource allocation tool, which we have implemented, is called TARF-tool. Using various planning algorithms based on first-in first-out (FIFO) fashion, last-in first-out (LIFO) fashion or heuristics, this tool is capable of solving both scheduling and transportational problems.

Using the TARF-tool's graphical user interface (GUI), you can specify tasks, which you want to schedule, available agents and resources, objective function to be optimized, run the TARF-tool solver and thus obtain an optimized schedule. In the last part of the thesis, two allocation scenarios demonstrate the features and functionality of the TARF-tool. Also presented there, the stress test results tell us, how the computation time of planners changes with increasing number of tasks.

Abstrakt

Distribuované plánování a rozvrhování hraje hlavní roli v mnoha průmyslových aplikacích, sahajících od multirobotických systémů, vojenských taktických operací až po výrobu a logistiku. V této diplomové práci představujeme a popisujeme multiagentní solver, který využívá technik sdílení, delegace a alokace úkolů na lokálně plánovaných zdrojích. Nástroj pro agentní alokaci úkolů a zdrojů, který jsme naimplementovali, se nazývá TARF-tool. Využívajíce různých plánovacích algoritmů založených na stylu first-in first-out (FIFO), last-in first-out (LIFO) či na heuristikách, je tento nástroj schopný řešit jak rozvrhovací tak i přepravovací problémy.

Použitím grafického uživatelského rozhraní nástroje TARF-tool, můžete specifikovat úkoly, které chcete naplánovat, dostupné agenty a zdroje, účelovou funkci, kterou chcete zoptimalizovat, spustit TARF-tool solver a získat tak optimalizovaný rozvrh. V poslední části diplomové práce demonstrují schopnosti a funkcionalitu nástroje TARF-tool dva alokační scénáře. Tam jsou také prezentovány výsledky zátěžové zkoušky, které nám říkají, jak se s přibývajícím počtem úkolů mění doba výpočtu plánovačů.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problems of distributed decision making and decentralized planning arise often in the real-world applications. Examples of such applications can be from military operations domain [1] [2]:

- Mission planning
- Area Surveillance
- Convoy protection
- Logistics

The most straightforward idea, how to solve such problems, is to use centralized planning algorithm. Centralized methods relies on one central planning entity, which requires data about all decentralized entities before the planning process. Such approach has several drawbacks. First, the centralized method assumes that all entities are able and willing to share their private information and plans. Second, centralized methods don't work in dynamic environment, where there is a requirement to be able to perform (real-time) replanning of plans. There may be a case, when such properties are not desirable.

Luckily, there is a second approach to distributed problems, a multi-agent decentralized approach with no central entity. The basic idea is that in cooperative environment, every agent manages its own plan and agents communicates between themselves. Such communication can contain for example negotiation about delegation of a task or about resource sharing.

In this master thesis, we will use this decentralized approach, specifically multi-agent planning algorithms based on task allocation and local resource planning in a cooperative environment.

Further in this chapter, in Section 1.1 you will find main objectives of this diploma thesis and the Section 1.2 shows the structure of the thesis.

1.1 Objective

The first objective of this thesis is to sufficiently describe and explain the theory needed for understanding the thesis. The main focus should be on general scheduling, multi-agent planning and task and resource allocation methods and algorithms.

The second task is to program a tool, which can be used to specify and solve distributed planning problems. This tool should use various allocation heuristics and algorithms and should be focused on scheduling and transportation problems. In order to achieve this, we should use the Alite toolkit.

The third objective is to design and implement a graphical tool for setting up agent-based allocation scenarios configurations. The tool will also be able to run the multi-agent problem solver and display a resulting schedule.

The last objective is to verify and demonstrate functionality and correctness of the implemented tools. This should be done firstly by testing the program and secondly by creating illustrative examples of allocation scenarios. There should be a scenario for both scheduling and transportation problem.

1.2 Structure of the thesis

The thesis is structured in the following way:

1. Chapter 1 introduces the topic of distributed planning and task and resource allocation problems. It describes the motivation behind, reveals objectives and shows structure of this thesis.

2. Chapter 2 contains necessary theory for understanding the thesis and proposes an architecture of the task allocation solver. The main topics are scheduling, task allocation and multiagent solver - its architecture and components.

3. In Chapter 3, there is a description of main components of the TARF-tool, which is the tool, that we have developed. Then it continues with description of implemented classes, which algorithms they use and what are good for. The chapter ends with a demonstration of the Graphical User Interface (GUI) of the TARF-tool.

4. Chapter 4 concerns validation and testing of the TARF-tool. There are two illustrative scenarios to demonstrate features and capability of the TARF-tool. Later, the chapter revolves around the stress test, its setting and results.

5. The last chapter (Chapter 5) summarizes what has been done in this thesis, what objectives have been accomplished and what are the consequences of that.

Chapter 2

Problem analysis and solution proposal

In this chapter, necessary theory from various science fields is explained, so the reader would understand the rest of the thesis.

As said in previous Chapter 1, this master thesis is about multi-agent planning and scheduling. But before we start explaining anything else, we would like to make just a small terminology note. The terms planning and scheduling are close, but not entirely the same. In scheduling often the tasks, which need to be performed, are already decided and set, and in practise tends to focus on algorithms for specific problem domains [10]. On the other hand, in planning one usually has to select the tasks, he want to do. The same holds for these terms in multiagent environment. Because of the overlap in the fields, we will not distinguish them and will use "planning" to refer to both planning and scheduling.

The chapter is divided in four sections: In Section 2.1, one approach of solving the multi-agent planning problems is presented, namely the task-sharing approach. It is shown, what types of agents are used in the solver and what is their role. Later in this section the problem of multi-agent planning is presented and then formalized.

Section 2.2 covers the introduction into the scheduling theory. It will cover formal description of scheduling problem, task parameters, task constraints (both general and specific), scheduling problem notation and optimality criterion.

Section 2.3 contains abstract architecture of multi-agent solver using task and resource allocation.

Section 2.4 presents the inner architecture of the allocation problem solver and its main components. The allocation mechanism based on Contract Net Protocol (CNP) is demonstrated later in the task allocation scenario.

2.1 Multi-agent Planning Problem

In this section, we will describe the task-sharing approach of solving the multiagent planning problems. It is shown, what types of agents are used in the solver and what is their role. Later in this section the problem of multiagent planning is presented and then formalized using the social welfare function.

The rest of this section is based on [15].

Multiagent planning solver Most of this subsection about multiagent planning solver was taken from [15].

As analyzed by Brafman and Domshlak in [5], the multiagent planning benefit from problems, where the domains of agents are relatively small and the agents are not so dependent on each other. The distributed planning and problem solving has been analyzed by Durfee [7]. A task-sharing approach was one of the discussed strategies. The main idea revolves around delegating tasks from busy agents to vacant agents.

The allocation problem is usually solved by contracting and negotiation techniques, which imply problems related to the resource allocation domain, e.g., cross booking, overbooking, backtracking, and others.

Generally, multiagent approaches show their strength in the domains, where the planning problem can be broken down into independent tasks. These tasks are usually allocated to different agents, and moreover these agents doesn't need to interact much. We will deal only with those types of problems, which can be decomposed like this. So we assume, that in the rest of this thesis, that tasks are independent, if we don't explicitly state otherwise.

The abstract multi-agent solver architecture based on the task-sharing approach from [15] is composed of three types of agents (see Figure 2.1).

1. Task agent: This agent is for preprocessing of the problem. It should use a domain-specific heuristic, generic ordering strategy, and randomized method.

2. Allocation agent: This agent is for problem decomposition into tasks and delegation of the tasks to resource agents. It maintains task allocation and result synthesis. This agent's strategies and algorithms are domain-independent.

3. Resource agent: This agent is for individual case-specific resource planning. In case of further decomposition, the task is handed over to another task agent.

The multi-agent solver, which is built of this architecture, usually contains one task agent, one allocation agent and more resource agents. These represent the distributed nature of the multiagent problem. This architecture can be used recurrently, one agent can has more abstract roles, or it can be parallelized (more abstract solvers operate over possibly overlapping agents). In such systems, concurrent interactions needs to be taken into account. The agents' communication uses interaction protocols, which are mostly built on Smith's contract net protocol (CNP) [12].

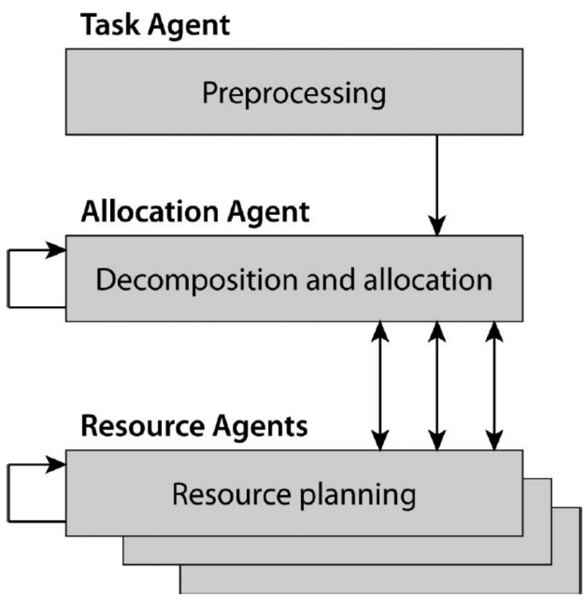


Figure 2.1: Abstract architecture of agent-based solver/planner from [15]

Multi-agent Problem The definition of the multi-agent planning problem cited from [15] follows.

The multi-agent solver uses the principles of problem decomposition and delegation to autonomous agents that solve parts of the problem individually. The overall solution is then obtained by merging the individual agents' results. The optimization based on CNP interactions in cooperative environments is usually described as utilitarian social welfare maximization [4]. Therefore, the abstract algorithm objective function can be defined as maximization of social welfare, which is as follows:

$$\mathrm{sw} = \sum_{a \in A} u_a \tag{2.1}$$

where $A = a_1, \ldots, a_n$ is the population of agents and u_a is the utility of agent a. In our case, the social welfare can be computed as a sum of resource agents ($R \subset A$) utilities that can be defined as follows:

$$u_a = \sum_{t \in T_a} (\mathrm{rew}(t) - \mathrm{cost}(t, a)) = \left(\sum_{t \in T_a} \mathrm{rew}(t) \right) - \mathrm{cost}(T_a) \tag{2.2}$$

where T_a is a set of tasks allocated to the agent $a \in R$, $\mathrm{rew}(t)$ is a reward for fulfilling task t, $\mathrm{cost}(t, a)$ is the cost of agent a to perform task t, and

$$\mathrm{cost}(T_a) = \sum_{t \in T_a} \mathrm{cost}(t, a) \tag{2.3}$$

is the cost of the overall plan of an agent. The total reward for fulfilling a set of all tasks T is as follows:

$$\operatorname{rew}(T) = \sum_{a \in R} \operatorname{rew}(T_a) = \sum_{a \in R} \sum_{t \in T_a} \operatorname{rew}(t) \tag{2.4}$$

so the social welfare can be expressed as follows:

$$\operatorname{sw} = \operatorname{rew}(T) - \sum_{a \in R} \operatorname{cost}(T_a) = \operatorname{rew}(T) - \sum_{t \in T} \operatorname{cost}(t, a) \tag{2.5}$$

Since we assume the same quality of task fulfilling by any agent, the reward $k = \operatorname{rew}(T)$ is not influenced by the allocation of tasks to the agents. We can derive social welfare as follows:

$$\operatorname{sw} = k - \sum_{t \in T} \operatorname{cost}(t, a) \tag{2.6}$$

As denoted earlier, the goal of CNP-based multi-agent optimization in cooperative environments is social welfare maximization. Given by (2.6), it is the same as minimization of solution cost, where $\operatorname{cost}(t, a)$ is evaluated by the resource agent a undertaking task t. The objective function of the abstract solver is then

$$\sum_{t \in T} \operatorname{cost}(t, a) \tag{2.7}$$

The task allocation stage of the solver searches for the best suitable mapping of the tasks T to the resource agents R that minimizes the objective function given by (2.7). We can define the goal of the allocation as finding such a partition P of the set of tasks T that

$$\arg\min_{P} \sum_{i=1}^{v} \operatorname{cost}(T_i) \tag{2.8}$$

where v is the number of resource agents, T_i is a subset of tasks allocated to the resource agent a_i , $\operatorname{cost}(T_i)$ is the cost of the overall plan of agent a_i performing T_i defined by (2.3), and

$$T_i \subseteq T, \bigcup_{i=1}^{v} T_i = T \tag{2.9}$$

$$\forall i, j : T_i \cap T_j = \emptyset \text{ iff } i \neq j. \tag{2.10}$$

2.2 Scheduling theory

In this section, a reader will find short introduction into scheduling theory from [8]. It will cover formal description of scheduling problem, task parameters, task constraints (both general and specific) and optimality criterion.

Scheduling problem Generally, the scheduling problem is problem of assignment of given tasks to the available resources in time. More formally, the input of the scheduling problem is defined as follows:

1. Set of n tasks $T = T_1, T_2, ..., T_n$
2. Constraints on the tasks T (such as release time, due time,...)
3. Set of m types of resources (processors, machines, employees,...) with capacities R_k, $P = \{P_1^1, ..., P_1^{R_1}, P_2^1, ..., P_2^{R_2}, ..., P_m^1, ..., P_m^{R_k}\}$

A result of the scheduling is a schedule, which determines which task is run on which resource and when. It is often depicted as a Gantt chart.

There are two types of scheduling problems. When the set of tasks is known, when executing the scheduling algorithm, it is called off-line scheduling. When on the other hand, the set of tasks is (partially) unknown and new tasks can be added to the set, it is the instance of on-line scheduling (one example can be scheduler in a operating system). In this thesis, we will deal only with the off-line scheduling.

There is one similar problem called planning. The main difference is that in scheduling, each task must be completed in contrast to deciding, which task will be scheduled and processed in the planning problem. One can say that the planning precedes scheduling.

Task parameters Every task is defined by its parameters. Figure 2.2 shows task parameters graphically. For the task T_j:

1. Processing time p_j is the amount of time needed to complete the task. This is the only parameter, that has to be necessarily specified for every task.
2. Release time r_j defines the earliest time of the task's start. The scheduled task cannot start before its release time. When the release time is not defined, it is the same as if the release time was set to zero.
3. Due date d_j is the time, in which task T_j should be completed.
4. Deadline $\tilde{d_j}$ is the time, in which task T_j has to be completed. The constraint defined by deadline is stronger than when using due time. When the deadline is not defined, it is the same as if the deadline was set to plus infinity.
5. Weight w_j specifies the importance of the task. The bigger the weight, the more important the task is (it has higher priority).

One can also define more parameters, such as cost of the task (in money).

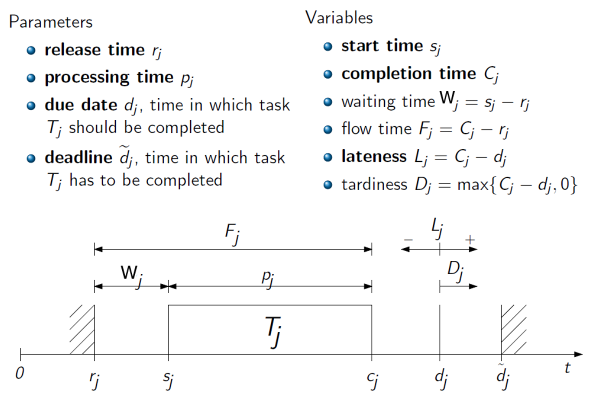

Parameters
release time r_j
processing time p_j
due date d_j, time in which task T_j should be completed
deadline $\tilde{d}_j$, time in which task T_j has to be completed
Variables
start time s_j
completion time C_j
waiting time $W_j = s_j - r_j$
flow time $F_j = C_j - r_j$
lateness $L_j = C_j - d_j$
tardiness $D_j = \max\{C_j - d_j, 0\}$
F_j
L_j
W_j
p_j
D_j
T_j
0
r_j
s_j
c_j
d_j
$\tilde{d}_j$
t

Figure 2.2: Task parameters and variables [8]

Tasks' constraints There are different kinds of tasks' constraints. There are general constraints, which must hold in every scheduling problem, and then there are specific constraints, that can be different for two scheduling problems.

Only two constraints belong to the general constraints:

1. Each task has to be sequential, meaning that each task is to be processed by at most one resource at a time.

2. Each resource is capable of processing at most one task at a time.

Below follows some of specific constraints:

1. Task T_i has to be processed during time interval $\langle r_i, \tilde{d}_i \rangle$, whenever both release time and deadline is defined.

2. When the precedence constraint is defined between T_i and T_j, i.e. $T_i < T_j$, then the processing of task T_j can't start before task T_i was completed. We also call this relationship a dependency or that the task T_i is predecessor of the task T_j.

3. If the scheduling problem is non-preemptive (preemption is not allowed), any task cannot be interrupted and resumed later (the task has to be performed as whole). In this thesis, we will focus only on non-preemptive scheduling.

Problem notation Classes of scheduling problems are specified in terms of a three-field classification $\alpha|\beta|\gamma$, where α specifies the resource characteristics, β specifies the job characteristics, and γ denotes the optimality criterion. We will introduce only a subset of possibilities of the notation from [6] and [8].

Resource characteristics α consists of two parts $\alpha = \alpha_1\alpha_2$. When $\alpha_1 = 1$, there is only one resource. If $\alpha_1 = P$, there are parallel identical resources, meaning they are multiple resources of the same kind. When you see $\alpha_1 = Q$, there are parallel uniform resources, which is similar to previous case but computation time of a task is inversely proportional to resource speed. The notation $\alpha_1 = PS$ means Project scheduling, which is a general case, where there are several resource types with capacities and general precedence constraints exist.

For $\alpha_1 = \emptyset$, there is arbitrary number of resources. If $\alpha_2 = k$, where k is a positive integer $1, 2, \ldots$, then α_2 denotes the number of resources. Notation $\alpha_2 = m, R$ means there are m resource types with capacities R (Project scheduling).

The task characteristics is specified by set β, which contains at most eight elements $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$. Value $\beta_1 = pmtn$ indicates, that preemption is allowed. Preemptions means, that executing of any task can be paused and resumed later. $\beta_2 = prec$ means there are precedence constraints between some tasks. If $\beta_2 = \emptyset$, the tasks are independent. Value $\beta_3 = r_j$ says, release dates may be specified for each task. If $\beta_5 = d_j$, due-date can be defined for every task, i.e. task t_j should end before due-date d_j.

The gamma letter γ denotes the optimality criterion. If $\gamma = \emptyset$, the scheduling problem is only a decision problem (we only care about existence of a valid schedule). The case $\gamma = C_{max}$ means, the goal is to minimize maximal completion time (makespan). For more optimality criteria, see paragraph *Optimality Criteria* later in this section.

Now a few examples: Notation $1|r_j|C_{max}$ denotes problem of finding schedule on one resource, for a set of tasks with given various release times $r_j \neq 0$ such that the maximal completion time is minimized. Earliest release time first is the optimal heuristic for this problem.

$P||C_{max}$ means problem with m identical resources and the length of schedule should be minimal.

Optimality Criteria The scheduling problem often defines an optimality criterion. The criterion, which has to be optimized (either maximized or minimized) in order to solve the problem. Rest of this section was taken from [6].

We denote the finishing time of task T_i by C_i, and the associated cost by $f_i(C_i)$. There are essentially two types of total cost functions

$$f_{max}(C) := \max\{f_i(C_i)|i = 1, ..., n\}$$

and

$$\sum f_i(C) := \sum_{i=1}^{n} f_i(C_i)$$

called bottleneck objectives and sum objectives, respectively. The scheduling problem is to find a feasible schedule which minimizes the total cost function.

The most common objective functions are the makespan $\max\{C_i|i = 1, ..., n\}$, total flow time $\sum\limits_{i=1}^{n} C_i$, and weighted (total) flow time $\sum\limits_{i=1}^{n} w_i C_i$

Other objective functions depend on due dates d_i which are associated with jobs J_i. We define for each job J_i:

$$
\begin{array}{rcl}
L_i := & C_i - d_i & \text{lateness} \\
E_i := & \max\{0, d_i - C_i\} & \text{earliness} \\
T_i := & \max\{0, C_i - d_i\} & \text{tardiness} \\
D_i := & |C_i - d_i| & \text{absolute deviation} \\
S_i := & (C_i - d_i)^2 & \text{squared deviation} \\
U_i := & \begin{cases} 0 & \text{if } C_i \leq d_i \\ 1 & \text{otherwise} \end{cases} & \text{unit penalty}
\end{array}
$$

2.3 Abstract Architecture

In this section, we will explain, how the multi-agent problem solver works and what parts it is made of and what algorithms it uses. The rest of this section is based on [15].

Abstract Algorithm The abstract algorithm from [15], which is based on the presented multi-agent solver attempting to minimize objective function defined by (2.7), is shown in Algorithm 1. As one can see from the abstract architecture shown in Figure 3.4, it consist of three phases as following.

1. Task preprocessing provided by the task agent is the first phase of the function `solve`. To increase the solver efficiency in the given domain, the domain-specific ordering heuristic is applied on the task set T.

2. The second phase is iteration over the whole task set and allocation performed by the allocation agent, which minimizes the insertion cost computed by resource agents (the `allocateCNP` function). As part of this iteration, the dynamic improvement based on cooperation of allocation agent and all resource agents takes place - the improvement strategy is applied to every resource agent after allocation of each task.

3. The third phase of the `solve` function is the final improvement of the solution. After allocation of all tasks the improvement strategy is executed by all resource agents.

The algorithm uses local optimization of a single-task insertion, which is followed by applying improvement strategy. Each iteration of the algorithm provides a greedy (order-dependent) task allocation with subsequent local optimization on resource agents.

For these computations, the resource agent uses a problem-dependent resource-planning heuristic. The functions for allocation are as follows.

1. *Insertion estimation cost*$^{estI}(t, a)$: The estimation of the cost of the task insertion. It represents the increase of the agent's a cost function caused by performing the task t.

2. *Insertion* $cost^{insert}(t,a)$: The real cost of the task insertion. This value is determined by adding a new task t to the plan of the agent a in the current state. It is the result of the planning algorithm of the resource agent.

The opposite functions used by improvement strategies are as follows.

1. *Removal estimation* $cost^{estR}(t,a)$: The estimation of the cost of the task removal. It represents the decrease of the agent's a cost function caused by deleting the task t from the plan.

2. *Removal* $cost^{remove}(t,a)$: The real cost of the task removal. This value is determined by removing the task t from the plan of agent a in the current state. It is the result of the planning algorithm of the resource agent.

The allocation in the CNP part of the Algorithm 1 is based on the determination of the winner agent. The winner of task t is a resource agent a with the lowest insertion cost; mathematically speaking,

$$winner = \arg\min_{a \in R} cost^{estI}(t,a). \tag{2.11}$$

The allocation agent allocates an unallocated task t, where $\forall a_i \in R : t \notin T_{a_i}$ to a winner agent a

$$allocate(T_a, t) \Rightarrow t \in T_a \tag{2.12}$$

given that local plan of agent a exists and the agent a is able to complete this task using the plan for the cost estimation $cost^{estI}(t,a)$ used in (2.11).

2.4 Inner architecture of the allocation solver

In this section, we present the inner architecture of the allocation problem solver and its main components. The allocation mechanism based on Contract Net Protocol (CNP) is demonstrated later in the task allocation scenario. We have used the Alite toolkit [3] mainly for the communication between the agents and the CNP protocol implementation.

The allocation agent and every resource agent has a communicator object through which it communicates with other agents. The communicator object has an address.

Task base Each resource agent and the allocation agent has a task base. The task base uses the communicator for negotiating the task allocation using the CNP protocol. Specifically, at the beginning a resource agent registers the capability of doing a certain task type to its task base. The agent is then notified by the task base, if someone (some other agent) wants to invoke a task of the registered type.

2.4.1 Resource scenario

Task Task is an object representing general tasks. It has following fields:

- Task ID - automatically generated and is unique for every task
- Task type - type of the task
- Release time - earliest time of the task's start
- Duration - processing time of the task
- Due time - the time, when the task should be completed
- Priority - priority of the task
- Cost - cost of the task, can be interpreted for example as money

Resource plan base Each resource agent has control over typically one resource that can perform one or more types of tasks. For this purpose, each resource agent has a resource plan base, that manages planning of that resource. The plan base specifies the planning algorithm and an evaluation function to provide insertion and removal cost estimations. The resource plan base is sometimes called a "general" plan base.

Contract Net Protocol The Foundation for Intelligent Physical Agents (FIPA) is an international organization that developed the Contract Net Interaction Protocol (CNP). The UML diagram of CNP can be seen in Figure 2.3. The original description of the CNP from the official FIPA website follows [13]:

> In the contract net IP, one agent (the Initiator) takes the role of manager which wishes to have some task performed by one or more other agents (the Participants) and further wishes to optimise a function that characterizes the task. This characteristic is commonly expressed as the price, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc. For a given task, any number of the Participants may respond with a proposal; the rest must refuse. Negotiations then continue with the Participants that proposed.

Task allocation scenario We will now describe a task allocation scenario. At the beginning, resource agents register itself to their task bases and they register itself to the task types, they have resource for.

If a new task should be planned, the allocation agent invokes the task through the task base. The task base looks if there is a resource agent communicating on the same communicator. If there is not one, the task cannot be allocated on any resource. But

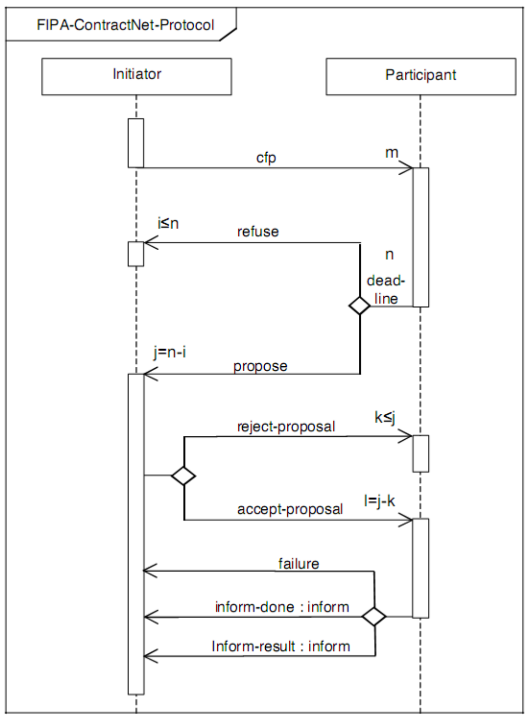


Figure 2.3: FIPA Contract Net Interaction Protocol [14]

if there is such resource agent, then the CNP protocol is started (this corresponds with calling the `allocateCNP` function from Algorithm 1).

The allocation agent plays the role of CNP Initiator and resource agents are Participants. The CNP starts with the allocation agent sending a message with call-for-proposal communicative act to all resource agents, that are registered for the task's type. All resource agents, which obtained a call-for-proposal message, prepare a response. Every resource agent gets from its plan base a cost estimation of inserting the task in its plan (in its planbase to more precise). A propose act with this cost estimate will be sent back to the allocation agent a.k.a. the Initiator. A resource agent can also send a reject act instead of proposal, meaning the agent doesn't want to or can't allocate the task.

The allocation agent waits for all proposals (for a limited time). Then it selects a winner, which is the resource agent with the lowest cost proposal. The allocation agents sends an accept-proposal act to the winner and reject-proposal acts to the other resource agents. The winning agent then has to allocate the task on its resource.

After the task is completed, the resource agent sends an inform-done act to the allocation agent to let it know, the task has been completed. If something goes wrong and the winner fails to complete the task, it is obliged to send a failure act to the allocation agent. Anyway, the allocation agent will know what happened with the task.

2.4.2 Logistical scenario

So far, we assumed general tasks, that are not tied to a specific location. But there can be scenarios, where position of resources and transportation between them plays an important role in the allocation problem. An examples of such domain, where the position aspect is crucial, are logistics or supply chain.

A logistic scenario is a scenario in which we have (apart from the general tasks) a set of transportational tasks. Each of the transportation tasks has defined start location, at which the task has to begin, and end location, where the task has to finish. One can see this logistical tasks as a task of transporting goods from a point A to point B. Capable of performing these logistic tasks, a special type of resource is defined, called transportational resource. Such resource can model many things from a truck to an airplane. Such logistic scenario is more complicated than a resource scenario, because beside the question "On which resource should we allocate this task?", there is a new question "Which path between the start and the end location should we take?" For purpose of modelling and simulating such scenarios, there are logistic (or transportational) variants of previously defined objects, such as logistic plan bases and logistic tasks.

Logistic task Logistic task can be viewed as an extension of general task. In contrast to a general task, logistic task has two additional parameters - start node and end node. The node is meant as a vertex in the planning problem roadmap. The logistic task should then be understood as a transportation task to deliver some object(s) from start node to end node. One more thing, that is different from general task, is computing of the processing time. The processing time of the logistic task is not an unchanging constant (as it is by general tasks), but it is dependent on the scheduled resource and on the last task on that resource. The resulting task duration can vary depending on how fast the resource can transport itself from the current location (or end node location of the resource's last task) to the start node of the new task.

Logistic plan base Logistic plan base is an extension of resource plan base. Logistic plan base is plan base for transportational resources such as truck, ship or plane. Unlike general plan base, the logistic plan base has one additional parameter - an initial position (initial node). This can impersonate for example a parking depot of a truck or a hangar of a plane. The second difference is, as mentioned in the previous paragraph, the necessity of allocating extra time to transport the resource to the start node in the first place, before the resource can perform the core logistic task.

Algorithm 1 The abstract algorithm of a multi-agent solver [15]

Input: Set of tasks T, set of resource agents R
Output: T allocated on R and local plans of resource agents exits

```
1: function SOLVE($T, R$)
2:     apply ordering heuristic on $T$
3:     for all $t : T$ do
4:         ALLOCATECNP($t$,$R$)
5:         if allocation not succesful then
6:             exit with failure or
7:             mark $t$ as not allocated and continue
8:         end if
9:         for all $a : R$ do
10:            apply dynamic improvement strategy
11:        end for
12:    end for
13:    for all $a : R$ do
14:        apply final improvement strategy
15:    end for
16: end function

17: function ALLOCATECNP($t, R$)
18:    for all $a : R$ do
19:        find winner with the lowest insertion estimation of $t$
20:    end for
21:    if winner is found then
22:        assign $t$ to the winner
23:    else
24:        allocation not successful
25:    end if
26: end function
```

Chapter 3

Implementation

The second objective of this thesis was to implement selected allocation heuristics and algorithms using Alite toolkit [3] with focus to scheduling and transportation problems. In order to complete this objective, we have developed an agent-based task and resource allocation tool, called TARF-tool. The TARF-tool was implemented using the abstract architecture, described in the Chapter 2. This chapter describes parts and mechanisms of the TARF-tool.

This chapter consist of three sections. In Section 3.1, our implementation of the abstract architecture from previous chapter is presented. The section revolves mainly around the implementation of the three types of agent, specially the resource agent and his main components, such as plan base, planner and evaluator.

In Section 3.2, important classes of the TARF-tool are shown. The section is composed of implementations of planners, heuristics and evaluators. Finally Section 3.3 describes parts of the Graphical User Interface (GUI) called the Configurator. It also shows how the Configurator looks like, what windows and tabs it consist of and how to work with them.

3.1 Architecture

In the Chapter 2, we presented the abstract architecture of a multi-agent solver. Our TARF-tool uses this architecture. Basic building blocks of the architecture are three types of agents - a task agent, an allocation agent and a resource agent. We will now present a high-level overview of our implementation of this architecture.

Agents There is one task agent in the TARF-tool. Our task agent provides only first-in-first-out (FIFO) task sorting and then successively invokes every task using the allocation agent. The allocation agent is present only in one instance. In the source code, the allocation agent is in the class named `TaskInvokerAgent`. The allocation agent manages delegation of the tasks to resource agents. Through its task base, it initiated the Contract Net Protocol (CNP), to allocate a task to the most suitable resource agent. The allocation agent controls no resource and it does no task decomposition. There are no solution improvement strategies applied in any of the agents. The most important and most complex type of agent is the resource agent. Every resource agent manages typically one resource, which can perform one or more task types. As it was said in previous chapter, every resource agent has its plan base, which takes care of the planning on the resource. This plan base contains two important objects - a planner of the resource and an evaluator.

Planner The planner object is an implementation of a planning algorithm and holds the resource plan itself too. Various planners exist, the most important implementations are first-in-first-out (FIFO) planner, last-in-first-out (LIFO) planner and heuristic planner. More about different implementations of planners in next section.

Heuristic The heuristic planner is little more complex and has one speciality, so it needs to be explained more here. Inside the heuristic planner is an ordered list (priority queue), which is sorted according to a heuristic (sorting policy). This heuristic provides the information, which tasks should be scheduled first and which tasks last. Some examples of heuristics: earliest release time first, shortest duration first. More thorough description of the heuristic awaits in next section.

Evaluator An evaluator is something, that evaluates a plan. It is a function, which assigns a real value to the plan (which is inside a planner). The `Evaluator` 's plan-evaluation method is called, when the plan base needs to compute insertion or removal cost estimate. Some of the evaluators are maximal-completion-time (also called makespan) or sum-of-the-waiting-times. An in-depth description of various evaluators will follow in the next section.

3.2 Implemented classes

In this section, we will present the most important classes in the TARF-tool along with some of their implementation details. Most of the space is devoted to implementations of the planner, comparator and evaluator.

3.2.1 Planner

A planner object, as described in the previous section, contains the plan itself and provides also methods to manipulate with it. A planner is represented by the `TarfPlan` interface in the TARF-tool. The class `java.util.TreeSet` is used in all implementations to store the plan with scheduled tasks. There are multiple implementations of the `TarfPlan` interface:

Every planner has inside a list of tasks called *ordered list*, which is used to store the order of the tasks, to be able to reproduce the desired order `TarfPlan.remove(t)` method. In every planner, the method works in the following way: first, the given task is removed, and second the whole plan is deleted and replanned using the ordered list, in order to possibly schedule some tasks earlier than they were before the removal.

FIFOPlan Simple implementation of `TarfPlan` interface. `FIFOPlan` allocates tasks in the first-in first-out (FIFO) fashion. This means that a new task is scheduled at the end of the plan. It provides operation insert in $O(d^2)$ and operation remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one).

It has an advantage over `LIFOPlan` and `HeuristicPlan` that it is stable, meaning the addition of new task, won't change current plan. If immutability of plan is needed, the FIFO order isn't necessary and smaller completion time (C_{max}) is preferred, one can try EarliestGapFirstPlan.

LIFOPlan Simple implementation of `TarfPlan` interface. `LIFOPlan` allocates tasks in the last-in first-out (LIFO) fashion. This means that a new task is scheduled at the beginning of the plan and rest of tasks is rescheduled after that. It provides operations insert and remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one).

EarliestGapFirstPlan The implementation of `TarfPlan` interface, which tries to fill empty gaps in the scheduled plan by trying to insert new tasks there and, if not possible, at the end of the plan. It is a variation of the FIFOPlan. As the FIFOPlan, it has an advantage over the `LIFOPlan` and the `HeuristicPlan` that it is stable, meaning the addition of new task, won't change current plan. But it has an advantage over the FIFOPlan that in some cases it has lower completion time (makespan - Cmax). It provides operations insert and remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one).

HeuristicsPlan The implementation of `TarfPlan` interface, which allocates tasks according to a heuristics (an implementation of `Comparator<TarfTask>`). It provides operations insert and remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one). Note, that the heuristic function doesn't appear in the asymptotic estimate.

3.2.2 Logistic Planner

Apart from normal resource planners, logistic variants of the planners exist. They are usually work in the same way as their resource planner counterparts. Logistic planners has to implement the `LogisticPlan` interface.

In all implementations, the insert task function goes as follows. The earliest possible start time of the task is computed (e.g. maximum of completion time of plan and release time of the task in the `FIFOLogisticPlan`). From this time the transportation resource moves itself to the start destination (start node) of the logistic task. After its arrival to the start location, it begins the logistic task to the end location (end node). For finding the shortest path in the roadmap, we use the A-star algorithm.

Individual description of the implementing classes follows:

FIFOLogisticPlan Simple implementation of `TarfPlan` interface. `FIFOLogisticPlan` allocates tasks in the first-in first-out (FIFO) fashion. This means that a new task is scheduled at the end of the plan. It provides operation insert in $O(d^2)$ and operation remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one).

LIFOLogisticPlan Simple implementation of `TarfPlan` interface. `LIFOLogisticPlan` allocates tasks in the last-in first-out (LIFO) fashion. This means that a new task is scheduled at the beginning of the plan and rest of tasks is rescheduled after that. It provides operations insert and remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one).

HeuristicsLogisticPlan The implementation of `TarfPlan` interface, which allocates tasks according to a heuristics. It provides operations insert and remove in $O(nd^2)$, where n is number of tasks in the plan and d is number of successors of inserted task (which is at most all tasks minus one). Note, that the heuristic function doesn't appear in the asymptotic estimate.

3.2.3 Heuristic

The `Heuristic` interface is used by the `HeuristicsPlan` and `HeuristicsLogisticPlan` as a heuristic to sort the tasks. The `Heuristic` implementations correspond to classical scheduling heuristics (as appears for example in [6]). In the enclosed source codes, the `Heuristic` implementations are called `Comparator`s (because they are implemented as a Java `Comparator`). There are several `Heuristic`s implemented:

EarliestDueTimeFirstComparator The implementation of `Comparator` , which schedules task with earliest due time first. This is optimal heuristic (sorting) for scheduling problems $1|d_j|C_{max}$, $1|d_j|L_{max}$ and recommended heuristic for the scheduling problem $1|d_j|\sum U_j$.

EarliestLatestStartTimeFirstComparator The implementation of `Comparator` , which schedules task with earliest latest-start time first. Latest start time is computed as task's duration minus due time and represents latest start time in order not to be late. The task, which has this value lowest, is scheduled first.

EarliestPossibleCompletionTimeFirstComparator The implementation of `Comparator` , which schedules task with earliest possible completion time first. Earliest possible completion time is computed as task's release time plus its duration and represents completion time of the task, if it would be started on its release time.

EarliestReleaseTimeFirstComparator The implementation of `Comparator` , which schedules task with earliest release time first. This is optimal heuristic (sorting) for the scheduling problem $1|r_j|C_{max}$.

HighestCostFirstComparator The implementation of `Comparator` , which schedules task with highest cost first.

HighestPriorityFirstComparator The implementation of `Comparator` , which schedules task with highest priority first. `HighestPriorityFirstComparator` can be given a second comparator, which would compare two tasks in case of equality of priorities. Default second comparator is `EarliestReleaseTimeFirstComparator`.

LeastSlackTimeFirstComparator The implementation of `Comparator` , which schedules task with least slack time first. Slack time is the amount of time left after a task if the task was started at its release time. Slack time is computed as task's due time minus its release time minus its duration. This heuristic is also known as Least laxity first. This is recommended heuristic for the scheduling problem $1|d_j|\sum D_j$.

LongestDurationFirstComparator The implementation of `Comparator` , which schedules task with longest duration first.

LowestCostFirstComparator The implementation of `Comparator` , which schedules task with lowest cost first.

ShortestDurationFirstComparator The implementation of `Comparator` , which schedules task with shortest duration first. This is optimal heuristic (sorting) for the scheduling problems $1||\sum C_j$ and $1||\sum W_j$ and recommended heuristic for the scheduling problem $1|d_j|\sum L_j$.

ShortestPriorityWeightedDurationFirstComparator The implementation of `Comparator` , which schedules task with shortest weighted duration first. The priority weighted duration is computed as task's duration multiplied by its priority. In another words, first will be scheduled short high-priority tasks. This is optimal heuristic (sorting) for problem $1||\sum w_j C_j$ and for the sum of priority weighted waiting times criteria (problem $1||\sum w_j W_j$).

3.2.4 Evaluator

Another important interface in the TARF-tool is the `Evaluator` interface. `Evaluator` is an interface for evaluating a `TarfPlan` according to a criteria function defined by the `Evaluator` implementation. This evaluation function is called, when a plan base needs to compute insertion or removal cost estimate. Below is a list of all implementations of `Evaluator` interface.

MaxCompletionTimeEvaluator The implementation of `Evaluator` interface, which returns end time of `TarfPlan`'s last scheduled task. The heuristic, which minimises this criterion (when not dealing with due times and precedence relations, i.e. scheduling problem $1|r_j|C_{max}$), is earliest release time first (implemented in `EarliestReleaseTimeFirstComparator`).

MaxLatenessEvaluator The implementation of `Evaluator` interface, which returns max of lateness of `TarfPlan`'s scheduled tasks. Lateness is defined as difference between scheduled end time of the task and task's due time. The heuristic, which minimises this criterion (when there are no release times and no precedence relations defined, i.e. scheduling problem $1|d_j|L_max$), is earliest due time first (implemented in `EarliestDueTimeFirstComparator`).

MaxTardinessEvaluator The implementation of `Evaluator` interface, which returns max of tardiness of `TarfPlan`'s scheduled tasks. Tardiness is defined as difference between scheduled end time of the task and task's due time or zero, if this difference is smaller than zero. The heuristic, which (along with slight modification of the scheduling algorithm) minimises this criterion (when there are no release times and no precedence relations defined, i.e. scheduling problem $1|d_j|T_{max}$), is earliest due time first (implemented in `EarliestDueTimeFirstComparator`).

NumberOfTasksEvaluator The implementation of `Evaluator` interface, which returns number of scheduled tasks in the `TarfPlan`. The recommended planning algorithm to minimize this criterion is FIFO (in class `FIFOPlan`), since it has fastest implementation of insert function and the order of the tasks have no effect on total number of tasks.

NumberOfLateTasksEvaluator The implementation of `Evaluator` interface, which returns number of late tasks in the `TarfPlan`, i.e. tasks, whose ends are scheduled after their due times. It also called unit penalty criterion. The heuristic, which (along with slight modification of the scheduling algorithm) minimises this criterion (when there are no release times and no precedence relations defined, i.e. scheduling problem $1|d_j|\sum U_j$), is earliest due time first (implemented in `EarliestDueTimeFirstComparator`).

SumOfWaitingTimesEvaluator The implementation of `Evaluator` interface, which returns sum of waiting times of `TarfPlan`'s scheduled tasks. The optimal heuristic for the scheduling problem $1||\sum W_j$ (i.e. no release times and no precedence relations defined) is called shortest processing time first (implemented in `ShortestDurationFirstComparator`).

The implementation of `Evaluator` interface, which returns sum of waiting times of `TarfPlan`'s scheduled tasks. The heuristic, which minimises sum of waiting time (when not dealing with release and due times), is called shortest processing time first (implemented in `ShortestDurationFirstComparator`).

SumOfPriorityWeightedWaitingTimesEvaluator The implementation of `Evaluator` interface, which returns sum of priority-weighted waiting times of `TarfPlan`'s scheduled tasks. Priority weighted waiting time is computed as waiting time divided by priority. The biggest contribution in the evaluation is made by important tasks that wait a long time. The optimal heuristic for the scheduling problem $1||\sum w_jC_j$ (i.e. no release times and no precedence relations defined) is called weighted shortest processing time first (implemented in `ShortestPriorityWeightedDurationFirstComparator`).

SumOfEarlinessEvaluator The implementation of `Evaluator` interface, which returns sum of earliness of `TarfPlan`'s scheduled tasks. Earliness is defined as difference between task's due time and scheduled end time.

SumOfLatenessEvaluator The implementation of `Evaluator` interface, which returns sum of lateness of `TarfPlan`'s scheduled tasks. Lateness is defined as difference between scheduled end time of the task and task's due time. The recommended heuristic (sub-optimal, but works well) for the scheduling problem $1||\sum L_j$ (i.e. no release times and no precedence relations defined) is called shortest processing time first (implemented in `ShortestDurationFirstComparator`).

SumOfImpunctualnessEvaluator The implementation of `Evaluator` interface, which returns sum of impunctualness of `TarfPlan`'s scheduled tasks. Impunctualness is defined as absolute value of difference between scheduled end time of the task and task's due time. In another words, task has impunctualness of zero if (and only if) it ends just in time of its due time. Impunctualness is also called an absolute deviation criterion. The heuristic, which is not truly minimising this criterion, but works pretty well in the reality (when there are no release times and no precedence relations defined, i.e. scheduling problem $1|d_j|\sum D_j$), is least slack time first (implemented in `LeastSlackTimeFirstComparator`).

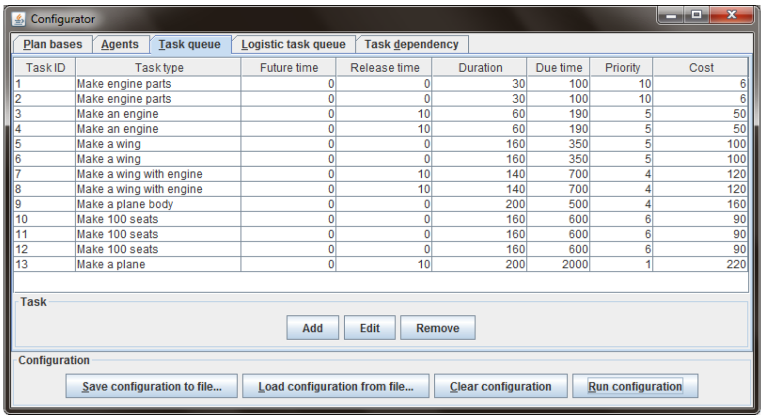
SumOfTaskCostsEvaluator The implementation of `Evaluator` interface, which returns sum of tasks' costs in the `TarfPlan`. The recommended planning algorithm to minimize this criterion is FIFO (in class `FIFOPlan`), since it has fastest implementation of insert function and the order of the tasks have no effect on total cost of the tasks.

SumOfTaskDurationsEvaluator The implementation of `Evaluator` interface, which returns sum of tasks' durations in the `TarfPlan`. The recommended planning algorithm to minimize this criterion is FIFO (in class `FIFOPlan`), since it has fastest implementation of insert function and the order of the tasks have no effect on sum of durations of the tasks.

3.3 Configurator - Graphical User Interface

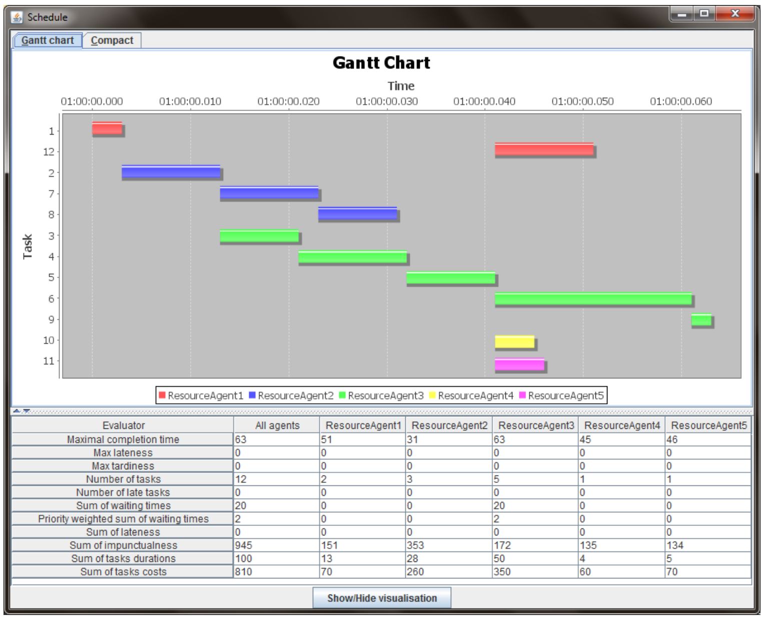
To be able to use the TARF-tool as a tool and not only as a Java library, we had to implement a way user can input his/her data and see the results. Because the approach with command line commands and parameters would be too complicated, we decide to create a Graphical User Interface (GUI). Our GUI is called Configurator, because via the Configurator the user can configure every parameter of his/her scheduling problem (scenario) and then can run the multi-agent solver on the scenario and see the found schedule.

The Configurator GUI uses three windows: the Configuration window (Figure 3.1), the Schedule window (Figure 3.2) and the Map window (Figure 3.8). First, we will start with the most important window - the Configuration window.



Task ID	Task type	Future time	Release time	Duration	Due time	Priority	Cost
1	Make engine parts	0	0	30	100	10	6
2	Make engine parts	0	0	30	100	10	6
3	Make an engine	0	10	60	190	5	50
4	Make an engine	0	10	60	190	5	50
5	Make a wing	0	0	160	350	5	100
6	Make a wing	0	0	160	350	5	100
7	Make a wing with engine	0	10	140	700	4	120
8	Make a wing with engine	0	10	140	700	4	120
9	Make a plane body	0	0	200	500	4	160
10	Make 100 seats	0	0	160	600	6	90
11	Make 100 seats	0	0	160	600	6	90
12	Make 100 seats	0	0	160	600	6	90
13	Make a plane	0	10	200	2000	1	220

Figure 3.1: The Task queue tab of the Configurator GUI



Evaluator	All agents	ResourceAgent1	ResourceAgent2	ResourceAgent3	ResourceAgent4	ResourceAgent5
Maximal completion time	63	51	31	63	45	46
Max lateness	0	0	0	0	0	0
Max tardiness	0	0	0	0	0	0
Number of tasks	12	2	3	5	1	1
Number of late tasks	0	0	0	0	0	0
Sum of waiting times	20	0	0	20	0	0
Priority weighted sum of waiting times	2	0	0	2	0	0
Sum of lateness	0	0	0	0	0	0
Sum of impunctualness	945	151	353	172	135	134
Sum of tasks durations	100	13	28	50	4	5
Sum of tasks costs	810	70	260	350	60	70

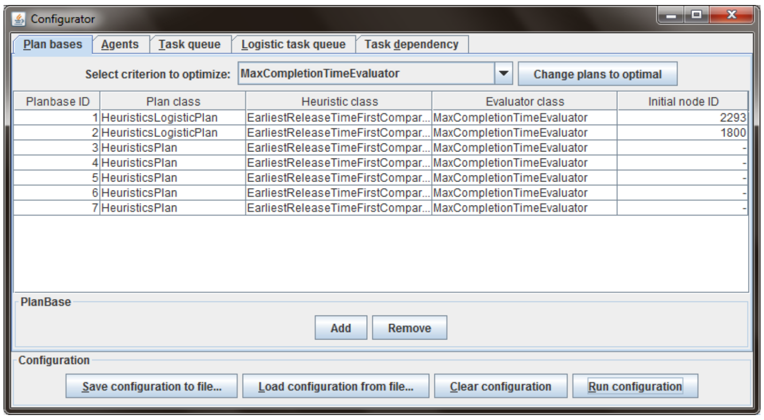
Figure 3.2: A Gantt chart visualising found schedule

When the Configurator GUI is started, the Configuration window is shown to a user. Main purpose of the configuration window is to enable the user to input his/her scheduling problem and set its parameters. In the lower part of the application window, there is a button panel. Using these button, the user can:

- Save the current configuration to a file

- Load configuration from a file
- Clear the configuration to start with an empty one
- Run the configuration and display found schedule

Both saving current configuration to a file and loading from a file uses Java Object Serialization, so the created configuration files are not readable for humans. In the upper part of the Configuration window, there is a panel with tabs, which contains five tabs - the Plan bases tab, the Agents tab, the Task queue tab, the Logistic task queue tab and the Task dependency tab.



Select criterion to optimize: MaxCompletionTimeEvaluator

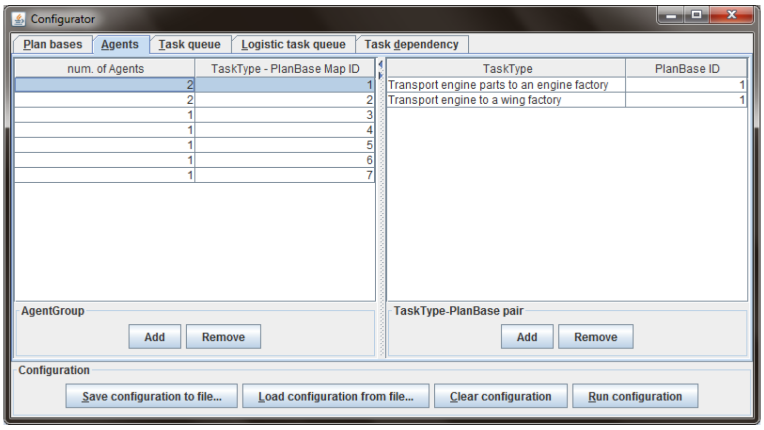
Planbase ID	Plan class	Heuristic class	Evaluator class	Initial node ID
1	HeuristicsLogisticPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	2293
2	HeuristicsLogisticPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	1800
3	HeuristicsPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	-
4	HeuristicsPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	-
5	HeuristicsPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	-
6	HeuristicsPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	-
7	HeuristicsPlan	EarliestReleaseTimeFirstCompar...	MaxCompletionTimeEvaluator	-

Figure 3.3: The Plan base tab of Configurator

Plan base tab In the Plan base tab (Figure 3.3), a user can create, edit and remove plan bases. The plan bases table has four columns:

- Plan base ID, which is automatically generated and is unique for every plan base
- Plan class, which specifies what planner the plan base use internally
- `Heuristic` class, which in case of `HeuristicPlan` defines, what heuristics should be used to sort the tasks
- `Evaluator` class that specify, what evaluator function should be used to evaluate plans

Agent tab The Agent tab itself (Figure 3.4) consists of two tables, one for manipulating with agents and the other one for displaying an associated map. The map takes types of tasks, that can the agent do and maps them to the corresponding plan bases that are used for scheduling the tasks. The agent table has two columns, the first specify number of agents with the same properties (same map) and the second tells an ID of a map.



num. of Agents	TaskType - PlanBase Map ID
2	1
2	2
1	3
1	4
1	5
1	6
1	7

TaskType	PlanBase ID
Transport engine parts to an engine factory	1
Transport engine to a wing factory	1

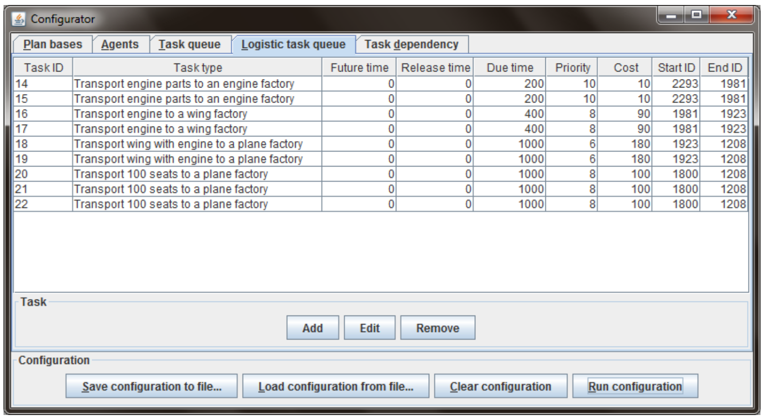
Figure 3.4: The Agent tab of the Configurator

Task queue tab In the Task queue tab (Figure 3.1), one can manage all tasks and their properties. It is possible to create, edit or remove tasks. A task has following properties:

- Task ID - automatically generated and is unique for every task
- Task type - type of the task
- Future time - time when the task will arrive to the task agent for scheduling (note that this feature has not yet been implemented)
- Release time - earliest time of the task's start
- Duration - duration of the task
- Due time - time, when the task should be completed
- Priority - priority of the task, lower number means higher importance
- Cost - cost of the task, can be interpreted e.g. as money.

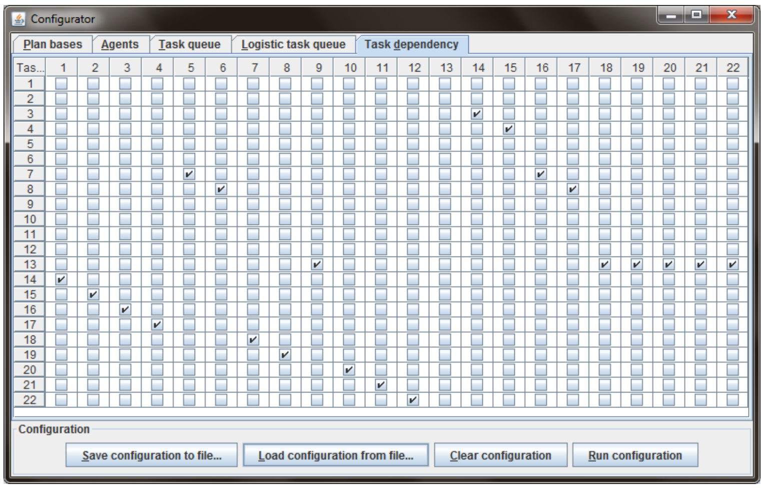
Logistic task queue tab The logistic task queue tab (Figure 3.5) is similar as the task queue tab. The difference is that the logistic task queue tab has two columns more for specifying a start and end node (more precisely node's ID) and lacks a column for a processing time of a task, since this property is computed from the scheduled path from start to end node.

Task dependency matrix tab In the task dependency tab (Figure 3.6), there is a table, which denotes dependencies between tasks (which are specified in the task queue tab and the logistic task queue tab). Every column and every row of the table represent a task, in each column and row header, there is the task's unique ID. There is a check mark



Task ID	Task type	Future time	Release time	Due time	Priority	Cost	Start ID	End ID
14	Transport engine parts to an engine factory	0	0	200	10	10	2293	1981
15	Transport engine parts to an engine factory	0	0	200	10	10	2293	1981
16	Transport engine to a wing factory	0	0	400	8	90	1981	1923
17	Transport engine to a wing factory	0	0	400	8	90	1981	1923
18	Transport wing with engine to a plane factory	0	0	1000	6	180	1923	1208
19	Transport wing with engine to a plane factory	0	0	1000	6	180	1923	1208
20	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208
21	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208
22	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208

Figure 3.5: The Logistic task queue tab of the Configurator GUI



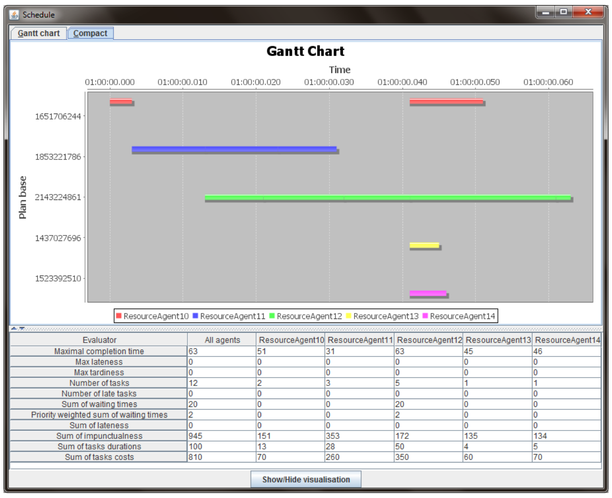
Tas...	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
2	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
3	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐
4	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐
5	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
6	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
7	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐
8	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐
9	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
10	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
11	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
12	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
13	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	✓	✓	✓	✓	✓
14	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
15	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
16	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
17	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
18	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
19	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
20	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
21	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
22	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Figure 3.6: The Task dependency tab of the Configurator GUI

(a tick) in the table in the row i and column j, if the row task T_i is dependent on the column task T_j.

The dependencies cannot be cyclic, otherwise the tasks would be unschedulable. If there are cyclic dependencies, the Configurator won't let the user run the simulation and will show an error window. Self-loops (when a task is dependent on itself), being cyclic dependencies itself, cannot be even entered in the table.

Schedule window After specifying the configuration and running it, there is a window for visualisation of the found schedule. There are two tabs: First is the Gantt chart (Figure 3.2), where each bar represents one task and every agent has its own colour of the tasks, and second one is the Compact chart (Figure 3.7), which offers a different, more compact representation of the schedule. Under both tabs, there is a table with statistics



Evaluator	All agents	ResourceAgent10	ResourceAgent11	ResourceAgent12	ResourceAgent13	ResourceAgent14
Maximal completion time	63	51	31	63	45	46
Max lateness	0	0	0	0	0	0
Max tardiness	0	0	0	0	0	0
Number of tasks	12	2	3	5	1	1
Number of late tasks	0	0	0	0	0	0
Sum of waiting times	20	0	0	20	0	0
Priority weighted sum of waiting times	2	0	0	2	0	0
Sum of lateness	0	0	0	0	0	0
Sum of impunctualness	945	151	353	172	135	134
Sum of tasks durations	100	13	28	50	4	5
Sum of tasks costs	810	70	260	350	60	70

Figure 3.7: The Compact chart for the found schedule

for every agent using different metrics (different Evaluators). Moreover, the first column shows statistics for all agents together. The JFreeChart library [11] was used for drawing the Gantt chart.

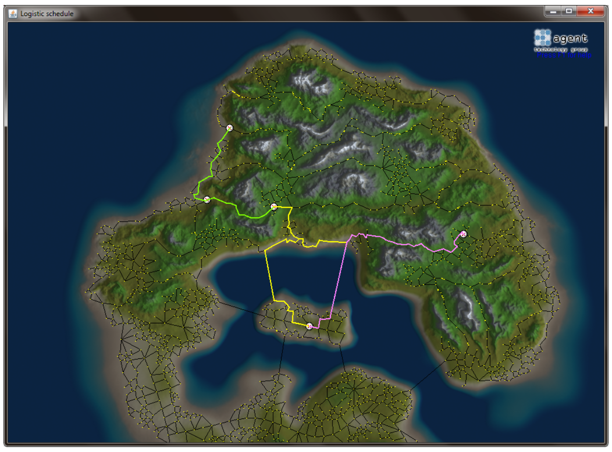


Figure 3.8: The Map Window of the Configurator GUI

Map window The Map window shows the map of the scenario. The Map window will show automatically if there is at least one logistic task scheduled. Yellow dots on the map represents nodes, black edges between them represents roads. White circle with small cross in it denotes a point of interest (start or end of a logistic task, starting location of logistic plan base). Colorful lines represents trajectories of logistical plan bases, each having one color.

The map can be zoomed and panned. Zooming can be done by mouse wheel and it magnifies area under mouse cursor. Panning can be done by drag and drop using right mouse button. A help screen can be displayed by pressing F1.

There is known issue with the Map window, that sometimes on some computers, the map can be drawn over another window. We don't know what is the cause of this bug, if it has something to do with a specific computer specification, a graphic card maybe, or it's caused by something else, but we at least figured out how to fix it. In case this bug occurs, just resize the bugged window and the window will repaint itself and everything will be back to normal.

Chapter 4

Testing and Validation

This chapter is devoted to testing and evaluation aspects of the TARF-tool. To ensure correctness of programmed TARF-tool, all planners, logistical planners, heuristics and evaluators are covered by unit tests. For this purpose, we have used Java unit testing framework JUnit 4. Only synchronized (one-threaded) simulation was implemented and tested.

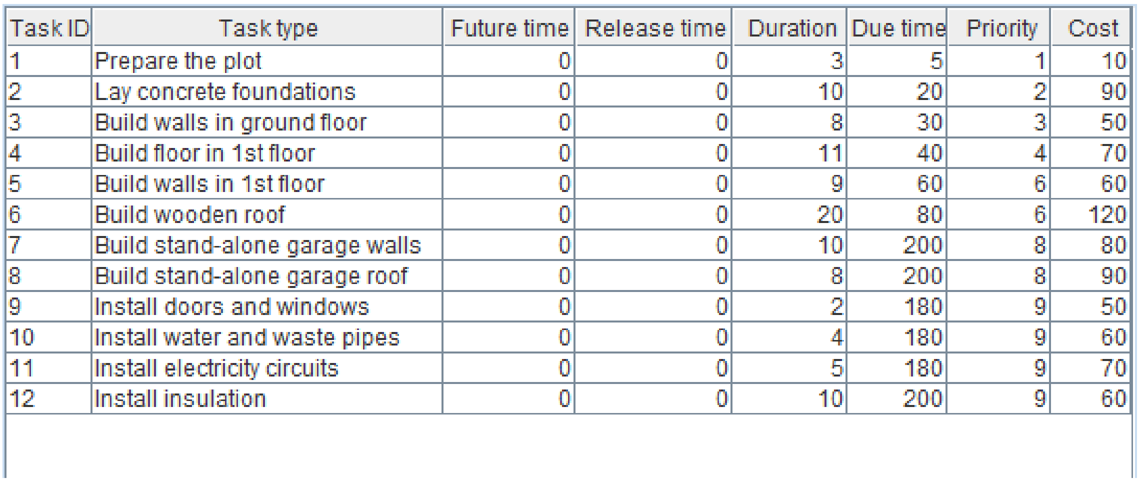
The chapter contains three sections. Firstly, Section 4.1 presents the house scenario, which contains only resource tasks (tasks with no relation to location). The scenario models how to build a two floor family house with a garage, which steps/tasks are needed and in which order.

Section 4.2 demonstrates a plane scenario, which is more complicated than the house scenario by taking locations and logistic into account. The objective to meet is to manufacture a plane from various parts which themselves can be composed of more components. There are various factories, where the components are made and there are also small and big trucks to transport the plane parts between factories.

Section 4.3 is dedicated to description of a stress test and its results. The stress test serves for the purpose of verify how much tasks the TARF-tool can handle and to measure speed of various planners.

4.1 Build a house scenario

To demonstrate the functions and features of the TARF-tool, we have created a resource scenario, which means the scenario has no logistic tasks. As a scenario, which we want to model, we have chosen a problem of building a family house. The main focus of this scenario is to verify the functionality of our program, not to capture the whole complex building process with all its details. Because of that the scenario is quite simplified. The house scenario file can be found in the enclosed program source files.



Task ID	Task type	Future time	Release time	Duration	Due time	Priority	Cost
1	Prepare the plot	0	0	3	5	1	10
2	Lay concrete foundations	0	0	10	20	2	90
3	Build walls in ground floor	0	0	8	30	3	50
4	Build floor in 1st floor	0	0	11	40	4	70
5	Build walls in 1st floor	0	0	9	60	6	60
6	Build wooden roof	0	0	20	80	6	120
7	Build stand-alone garage walls	0	0	10	200	8	80
8	Build stand-alone garage roof	0	0	8	200	8	90
9	Install doors and windows	0	0	2	180	9	50
10	Install water and waste pipes	0	0	4	180	9	60
11	Install electricity circuits	0	0	5	180	9	70
12	Install insulation	0	0	10	200	9	60

Figure 4.1: The definition of the resource tasks in house scenario

Setup The goal of this scenario is to build a two-floor family house with basic interior works finished and with stand-alone concrete garage. There are twelve resource tasks in this scenario. You can see full specification of the tasks in Figure 4.1.

The tasks ranges from preparing the plot, laying the foundations through building walls and floor in both ground floor and first floor to installing water and waste pipes and electricity. There are also two more independent tasks of building stand-alone concrete garage. An objective function, which is to be minimized is maximal lateness time, selected planner is heuristic plan with earliest release time first heuristic.

Every (resource) agent and its plan base represents a company or a person(s), which do one type of work. There are five agents (in parenthesis is their associated color): general site-workers (red), a company doing works with concrete (blue), carpenters (green), a plumber (yellow) and company that does the electricity work (pink).

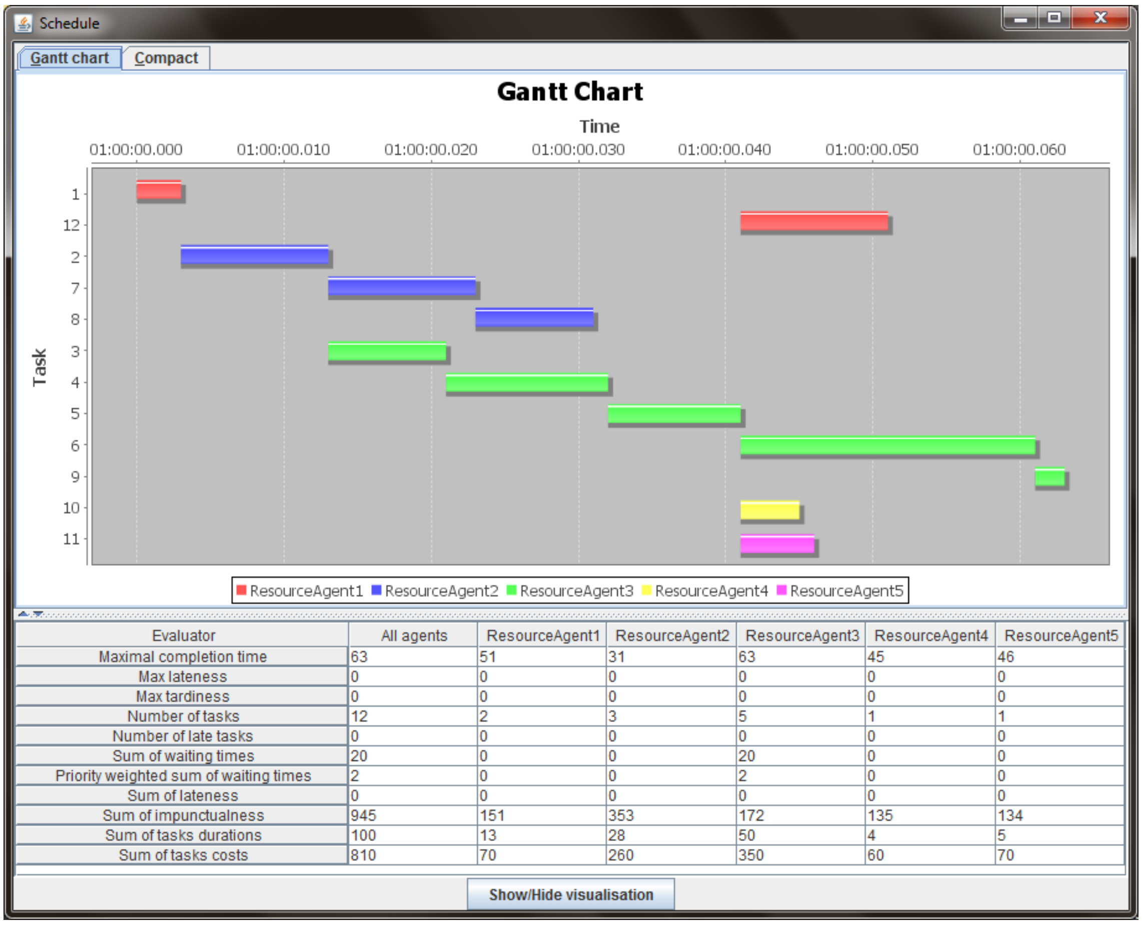
Results Found schedule with statistic numbers for the house scenario can be view in Figure 4.2. The schedule ends at time 63 and has no late tasks.

There is an interesting part of the plan, where a lot of tasks starts at time 41. This is because many tasks has building the walls in both floors as predecessors (they are dependent on it). So as soon as the all walls are finished, the interior tasks can be started.

4.2 Manufacture a plane scenario

The plane scenario is a representative of logistic scenarios, scenarios, where logistics and transportation between various places plays important part of the problem.

The plane scenario takes place on a big island, where multiple warehouses and factories are located. There are also multiple trucks at various locations on the island, that can be



Evaluator	All agents	ResourceAgent1	ResourceAgent2	ResourceAgent3	ResourceAgent4	ResourceAgent5
Maximal completion time	63	51	31	63	45	46
Max lateness	0	0	0	0	0	0
Max tardiness	0	0	0	0	0	0
Number of tasks	12	2	3	5	1	1
Number of late tasks	0	0	0	0	0	0
Sum of waiting times	20	0	0	20	0	0
Priority weighted sum of waiting times	2	0	0	2	0	0
Sum of lateness	0	0	0	0	0	0
Sum of impunctualness	945	151	353	172	135	134
Sum of tasks durations	100	13	28	50	4	5
Sum of tasks costs	810	70	260	350	60	70

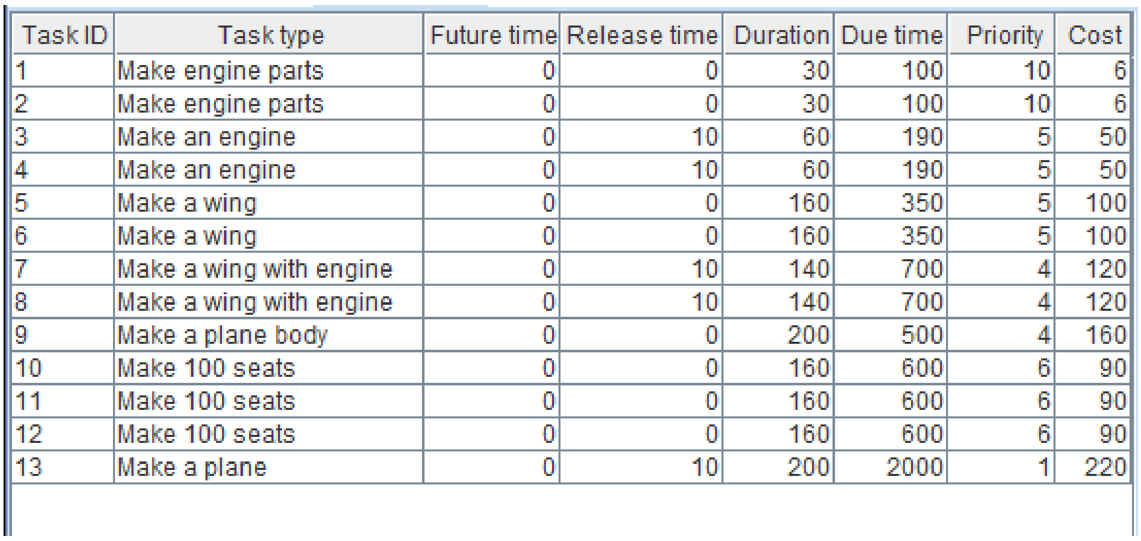
Figure 4.2: Planned schedule for the house scenario

used to transport goods. The major task is to build a plane. The model of the building process is again quite simplified, but it serves as demonstration. The island map and the roadmap was taken from the project I-Globe [9].

Setup There are several resource tasks in the plane scenario. The resource tasks are usually to manufacture or assembly a product.

To assemble a plane, one plane body, two wings with engines and 300 seats are required. Moreover, this assembly can be done only in a plane factory. A wing with an engine needs both a wing (without engine) and an engine. Both wing and wing with an engine are made in a wing factory. An engine is made out of engine parts and has to be assembled in the engine factory, but the engine parts themselves has to be made in an engine parts factory. The seat factory is a place to manufacture plane seats. Detailed description of the resource tasks can be found in Figure 4.3.

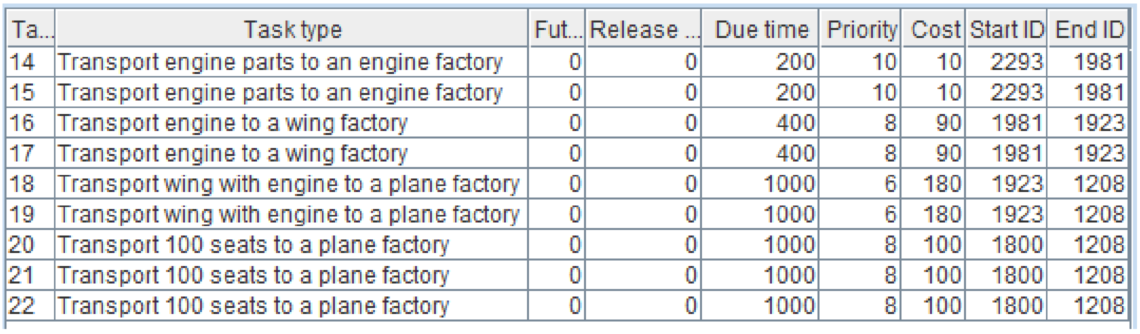
Every above mentioned factory is modelled as one agent with one plan base, which can



Task ID	Task type	Future time	Release time	Duration	Due time	Priority	Cost
1	Make engine parts	0	0	30	100	10	6
2	Make engine parts	0	0	30	100	10	6
3	Make an engine	0	10	60	190	5	50
4	Make an engine	0	10	60	190	5	50
5	Make a wing	0	0	160	350	5	100
6	Make a wing	0	0	160	350	5	100
7	Make a wing with engine	0	10	140	700	4	120
8	Make a wing with engine	0	10	140	700	4	120
9	Make a plane body	0	0	200	500	4	160
10	Make 100 seats	0	0	160	600	6	90
11	Make 100 seats	0	0	160	600	6	90
12	Make 100 seats	0	0	160	600	6	90
13	Make a plane	0	10	200	2000	1	220

Figure 4.3: The definition of the resource tasks in the plane scenario

perform tasks of the factory type (e.g. the wing factory can perform “Make a wing” and “Make a wing with engine” types).



Ta...	Task type	Fut...	Release ...	Due time	Priority	Cost	Start ID	End ID
14	Transport engine parts to an engine factory	0	0	200	10	10	2293	1981
15	Transport engine parts to an engine factory	0	0	200	10	10	2293	1981
16	Transport engine to a wing factory	0	0	400	8	90	1981	1923
17	Transport engine to a wing factory	0	0	400	8	90	1981	1923
18	Transport wing with engine to a plane factory	0	0	1000	6	180	1923	1208
19	Transport wing with engine to a plane factory	0	0	1000	6	180	1923	1208
20	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208
21	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208
22	Transport 100 seats to a plane factory	0	0	1000	8	100	1800	1208

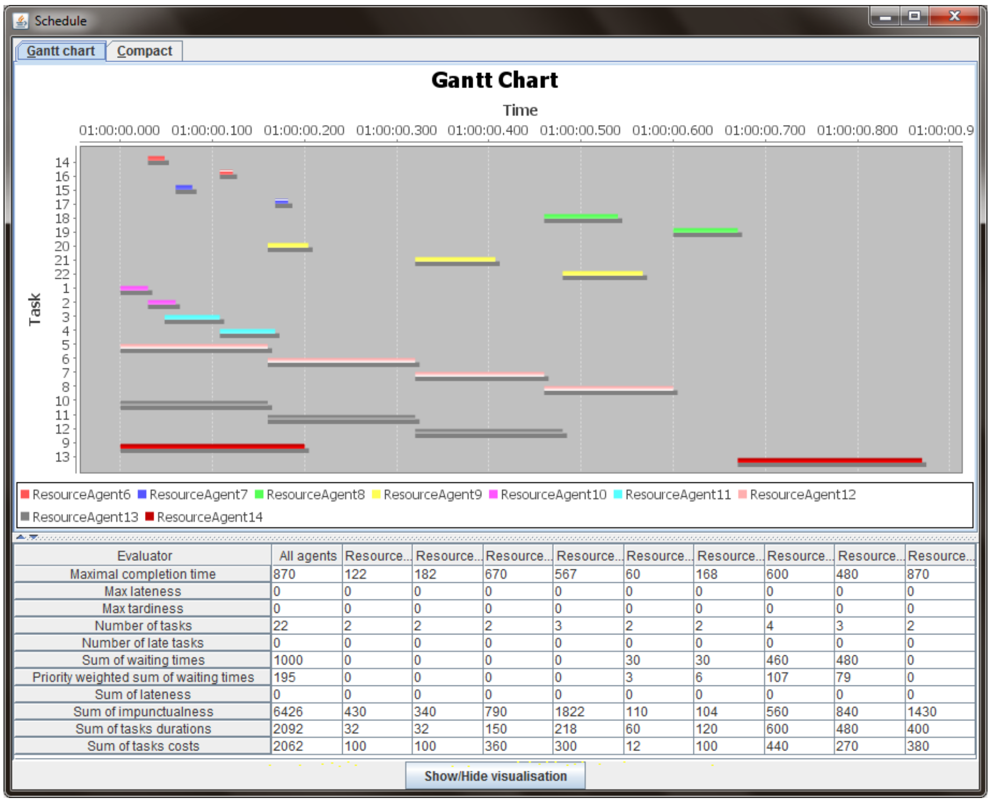
Figure 4.4: The definition of the logistical tasks in the plane scenario

Logistic tasks As is apparent, form the previous paragraphs, there are several logistic tasks to be made in order to produce one plane. There are tasks to transport engine parts to engine factory, a whole engine to wing factory or a wing to plane factory. Transportation of 100 seats from a seat factory to a plane factory is also a logistic task.

Dependency and precedence relations are done in such way that the transportation of an object waits for the object to be manufactured. Oppositely a production of the next product in the another factory has to wait until the transportation of the required goods ends. More detail information is presented in Figure 4.4.

There are two kinds of logistic plan bases - a small truck and a big truck. The small truck is used to transport engine parts and an engine. The big truck can transport everything else meaning a wing with engine, plane seats.

Objective function, which is to be minimized, is maximum completion time (makespan) and selected planner is a heuristic planner with earliest release time first heuristic.



Evaluator	All agents	Resource...	Resource...	Resource...	Resource...	Resource...	Resource...	Resource...	Resource...	Resource...
Maximal completion time	870	122	182	670	567	60	168	600	480	870
Max lateness	0	0	0	0	0	0	0	0	0	0
Max tardiness	0	0	0	0	0	0	0	0	0	0
Number of tasks	22	2	2	2	3	2	2	4	3	2
Number of late tasks	0	0	0	0	0	0	0	0	0	0
Sum of waiting times	1000	0	0	0	0	30	30	460	480	0
Priority weighted sum of waiting times	195	0	0	0	0	3	6	107	79	0
Sum of lateness	0	0	0	0	0	0	0	0	0	0
Sum of impunctualness	6426	430	340	790	1822	110	104	560	840	1430
Sum of tasks durations	2092	32	32	150	218	60	120	600	480	400
Sum of tasks costs	2062	100	100	360	300	12	100	440	270	380

Figure 4.5: Found schedule for the plane scenario

Results Found schedule for the plane scenario can be found in Figure 4.5. The schedule shows that the plane is manufactured at time 870 as the last task, which is logical since it has many predecessors and no successor. There are no late tasks.

In Figure 4.6, the logistic plan with all planned trajectories is shown. The green path belong to two identical small truck plan bases. The yellow and pink trajectories belong to big truck plan bases.

4.3 Stress scenario

To observe how the computation time of planners changes with an increasing number of tasks, we have constructed a stress test. We simply set up the scenario configuration and measured how long the TARF-tool needed to find the final schedule.

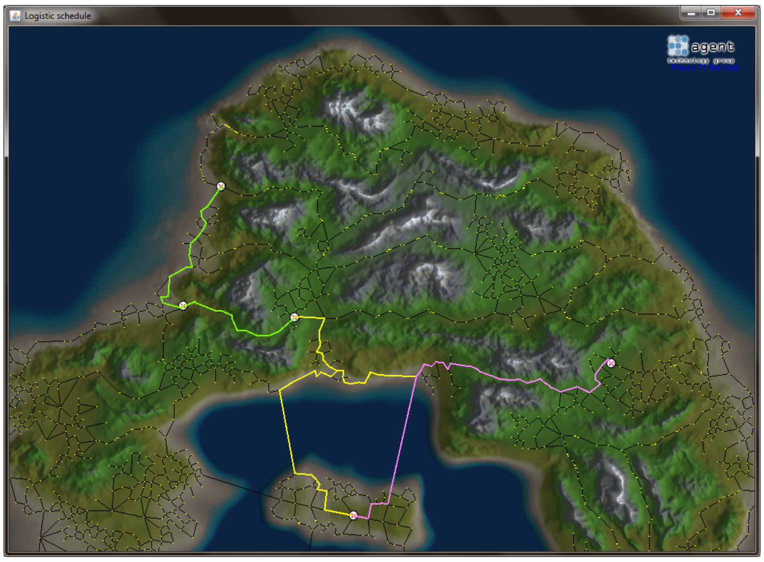


Figure 4.6: Found trajectories for the logistic plan bases for the plane scenario

Setup In Table 4.1, there is a definition of parameters of stress test. Most of them are maximal values of parameters, which are used when randomly generating tasks.

Table 4.1: Definition of parameters of stress test.

Parameter	Value
Max release time	100
Max duration	100
Max due time	1100
Max node ID	100
Number of repeats	10
Number of agents	2

Every node on the island map (a yellow dot on the map, usually on crossroad) has its ID. The parameter *Max node ID* is the highest node ID , that can be generated for logistic task start or end location. That means that in the stress test, all node IDs are in range 0 - 100.

There is one parameter of the test that is not in the Table 4.1. Parameter called Predecessor share (PS) is used when generating a random task. It says what percent from maximal possible number of task's predecessors the task actually will get as predecessor. The maximal number of task's predecessors is the biggest number of predecessors, such that there is no cyclic dependency in the task set.

For predecessor share $PS = 0$, all tasks have no predecessor. If predecessor share $PS = 1$, it means that if we would sort the tasks by increasing number of predecessors, the first task would have no predecessors, the second task would have one predecessor (in fact the first task would be this predecessor) and so on up to the last task, which would have all previous tasks as predecessors.

The stress test contains different configurations of a task set, which are given by two numbers - the number of tasks to be scheduled and the PS parameter. The number of tasks has four different values: 100, 200, 500 and 1000. The PS parameter takes either low or high value, low being 10 % and high being 90 %. Both logistical and non-logistical planners are tested. Resource (non-logistical) planners being FIFO, LIFO, Heuristic and Earliest gap first planner and logistic planners are FIFO logistic, LIFO logistic and Heuristic logistic planner. Each processing time has been measured ten times and then median was computed and saved into the table.

The stress test was done in testing environment specified by Table 4.2.

Table 4.2: Testing environment of the stress test.

Parameter	Value
CPU	Intel Core i3, 2x 2.53 GHz
Memory size (RAM)	4 GB
Operating system	Windows 7 Professional (64-bit)
Java version	JDK 1.7

Results The results are presented in two tables - Table 4.3 shows test result for resource planners and Table 4.4 shows test result for logistical planners. In both tables the columns represents a planning time of a planner and rows respond to one configuration of task set, which is given by number of tasks and the PS parameter, defining an amount of predecessors. Let's have a closer look on the resource planners.

Table 4.3: The stress test result for resource planners. PS denotes the predecessors share in percents.

		Planning time [s]			
# tasks	PS [%]	FIFOPlan	LIFOPlan	HeuristicsPlan	EarliestGapFirstPlan
100	10 %	0,071	0,043	0,032	0,042
100	90 %	0,024	0,019	0,018	0,020
200	10 %	0,030	0,075	0,045	0,028
200	90 %	0,128	0,109	0,093	0,077
500	10 %	0,204	0,999	0,325	0,194
500	90 %	1,995	1,874	1,824	1,804
1000	10 %	0,816	18,756	1,460	0,916
1000	90 %	16,094	17,850	16,440	16,108

Resource planners From the results in Table 4.3, one can see that times of the planners are very good even for the 1000 tasks, apart from few exceptions.

First exception is the row with 1000 tasks and PS = 90 %, this row contains rather slower times around 17 seconds. These times are not really usable for a real-time system, but they are still quite good. This drastic growth of processing times is most likely due to the high percentage of PS. Because after each successful task allocating, a planner has to notify all task's successors, that this task has been allocated and provide the successors with information about the task's scheduled end, so that the successors can update their release times. This also explains, why the even rows have higher times than the odd ones.

Another case of high processing time is in the column of the LIFO planner in the row with 1000 tasks with low (10%) PS value. This is caused by the implementation of the LIFO planner. The LIFO planner uses last-in first-out approach, which means that a new incoming task is scheduled in front of all scheduled tasks (as much as the precedence relations allow it). This basically means, that in the case of no predecessors, the new tasks is inserted at the beginning of the plan and the rest of the plan has to replanned. In this case the procedure is called 1000 times, because the test contains 1000 tasks. That's why we think it takes so long.

The table also tells us, the fastest planner is the FIFO planner, followed closely by the Earliest gap first planner, which makes sense, since it is a modification of the FIFO planner. On third place stands the Heuristic planner, that achieves a little bit slower times, however still usable. The last place goes to the LIFO planner.

Table 4.4: The stress test result for logistic planners. PS denotes the predecessors share in percents. Dash (-) means that the test run too long and was terminated.

		Planning time [s]		
# tasks	PS [%]	FIFOLogisticPlan	LIFOLogisticPlan	HeuristicsLogisticPlan
100	10 %	0,141	3,158	0,977
100	90 %	0,093	0,191	0,133
200	10 %	0,132	12,297	3,563
200	90 %	0,237	0,531	0,261
500	10 %	0,510	279,424	23,540
500	90 %	2,694	12,102	3,032
1000	10 %	1,533	-	-
1000	90 %	17,693	-	-

Logistic planners The Table 4.4, which displays stress test result for the logistic planners, is quite similar to the previous one in many things. As discussed above, rows with high PS value have again higher processing times because of the notifying of task's successors. The order of the logistic planners hasn't changed from the non-logistic ones. The FIFO logistic planner is the fastest, then Heuristic logistic planner and slow LIFO logistic planner sits at the tail.

One notable difference between the tables is the presence of dashes in the logistic table. The dash symbol in a table cell means that the particular test run too long and was terminated. As you can see from the table, dashes are present only in scenarios with 1000 tasks and only in columns of LIFO logistic planner and Heuristic logistic planner. Another interesting thing about the table is that the processing times are longer than in the resource Table 4.3. Both observations have a common explanation.

In a logistic task, there are parameters specifying the start and end node on the map. When scheduling a logistic task, the logistic planners have to first call a routine for finding the shortest path form the start node to end node, compute a duration of the task from the length of the path and after that schedule the task in the plan. The path-finding algorithm is quite fast, however when called a thousand times, it can make difference from not calling it in the non-logistic case. This effect is most visible in the case of LIFO logistic planner. Since the planner replans tasks a lot, the path-finding method is called a lot and the path-finding component starts to play significant role in the processing time.

Chapter 5

Conclusion

In this diploma thesis, we have studied the topic of multi-agent scheduling. We have also developed an agent-based task and resource allocation tool, which we have later tested. We have accomplished all four tasks of this thesis, which were described in Chapter 1.

Main contributions of this thesis are:

In Chapter 2 we have presented comprehensive introduction to theory of scheduling, multi-agent planning and scheduling. Later in the chapter, we have described an abstract architecture of multi-agent solver. In Section 2.4 we have proposed the inner architecture of the allocation problem solver. The allocation mechanism based on Contract Net Protocol (CNP) is demonstrated in the task allocation scenario.

Following the architecture from Chapter 2, we have implemented the task and resource allocation tool (TARF-tool). We have designed the program's architecture and implemented it in Java using Alite toolkit. The TARF-tool is capable of solving multi-agent scheduling and transportational problems. using selected allocation heuristics and algorithms, which were also implemented. More thorough description of the implementation of the tool is located in the Chapter 3.

For users to be able to use the TARF-tool, we have developed a Graphical User Interface (GUI). Our GUI is called Configurator, because via the Configurator users can configure every parameter of his/her scheduling problem (scenario), then run the multi-agent solver on the scenario and see the final schedule. Users can also save their scenario configuration into a file or load one. When the scenario is logistical, users can see a map of the schedule (planned trajectories). Everything about the Configurator can be found in Chapter 3, Section 3.3.

Another contribution is that we have tested our implemented TARF-tool to make sure, it works as it should. First, every heuristic, planner, logistic planner and evaluator is covered by unit tests. Second, we have built illustrative example scenarios to demonstrate features and functionality of the TARF-tool. The house scenario uses building a family house as a domain to demonstrate function of tasks, task dependency and agents. The plane scenario shows on the simplified example of manufacturing a plane, how to incorporate locations in the scenarios. Here, we have used advanced features of the TARF-tool such as logistic tasks, logistic plan bases and showing their trajectories on a map.

Third, to observe how the computation time of planners changes with increasing number of tasks, we have constructed a simple stress test. The results of these tests confirmed our expectation, that the more successors a task has, the longer time the insert task function needs. Another conclusion made from the test is that logistical planners are slower, because inside the insert function they have to run a path-finding algorithm. The fastest planner is the FIFO planner and its logistic variant. All about testing the TARF-tool is written in Chapter 4.

At this point, the TARF-tool is capable of solving some basic scheduling and transportational problems. There are of course always ways to improve it. One possible feature, which can be implemented would be ability to solve more complicated problems, for example a vehicle routing problem or another NP-hard problem.

Bibliography

[1] Agent Technology Center. *Tactical AGENTFLY* [online]. 2014. [cit. 10. 5. 2014]. Available from: <http://agents.cz/projects/agentfly/tactical>.

[2] Agent Technology Center. *AgentScout* [online]. 2014. [cit. 10. 5. 2014]. Available from: <http://agents.felk.cvut.cz/projects/agentscout>.

[3] Agent Technology Center. *Alite Homepage* [online]. 2014. [cit. 24. 3. 2014]. Available from: <http://jones.felk.cvut.cz/redmine/projects/alite/wiki>.

[4] ARROW, K. J. – SEN, A. K. – SUZUMURA, K. (Ed.). *Handbook of Social Choice and Welfare (Handbooks in Economics).* 1. North Holland, 1. edition, August . ISBN 0444829148.

[5] BRAFMAN, R. I. – DOMSHLAK, C. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *ICAPS*, s. 28–35, 2008.

[6] BRUCKER, P. *Scheduling Algorithms.* Springer, 2004. ISBN 9783540205241.

[7] DURFEE, E. H. Distributed problem solving and planning. In WEISS, G. (Ed.) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Intelligent Robotics and Autonomous Agents Series. MIT Press, 1999. Available from: <http://books.google.cz/books?id=JYcznFCN3xcC>. ISBN 9780262731317.

[8] HANZALEK, Z. – SUCHA, P. Scheduling. Combinatorial optimization - lecture slides, 2013. Available from: <https://moodle.dce.fel.cvut.cz/course/view.php?id=21>. [cit. 17. 4. 2014].

[9] KOMENDA, A. et al. I-Globe: Distributed Planning and Coordination of Mixed-initiative Activities. In *Proceedings of Knowledge Systems for Coalition Operations (KSCO 2009)*, 2009.

[10] National Aeronautics and Space Administration. *Workshop on Multiagent Planning and Scheduling* [online]. 2014. [cit. 3. 4. 2014]. Available from: <http://ai.jpl.nasa.gov/public/home/bclement/icaps05-workshop-map.html>.

[11] Object Refinery Limited. *JFreeChart* [online]. 2014. [cit. 24. 3. 2014]. Available from: <http://www.jfree.org/jfreechart/>.

[12] SMITH, R. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*. 1980, 29, 12, s. 1104–1113.

[13] The Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification* [online]. 2014. [cit. 12. 5. 2014]. Available from: <http://www.fipa.org/specs/fipa00029/SC00029H.html>.

[14] The Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol image* [online]. 2014. [cit. 12. 5. 2014]. Available from: <http://robotic-and-natural-language.googlecode.com/files/contract_net_protocol.PNG>.

[15] VOKRINEK, J. – KOMENDA, A. – PECHOUCEK, M. Abstract Architecture for Task-oriented Multi-agent Problem Solving. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*. January 2011, 41, 1, s. 31 –40. ISSN 1094-6977.

Appendix A

Source codes

The source codes of our allocation tool called TARF-tool, are enclosed on the CD.

The implementations of plans, planners, heuristics and evaluators are in the directory

- tarf-tool/src/main/java/cz/agents/tarftool/communication/plan/

The Main class is located in the

- tarf-tool/src/main/java/cz/agents/tarftool/Main.java

There are two sample scenarios packed with the program. You can find them in folder:

- tarf-tool/sample_configuration/

Mercurial repository The TARF-tool can be also pulled from the following mercurial repository link, where you substitute the word "USER" with your username:

`ssh://USER@smith.felk.cvut.cz//data/hg/incubator/d3cos-tarf`

How do I run it? To run the TARF-tool GUI, simply run class called "Main" located in the root of package "cz.agents.tarftool".