PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL CAUÊ CARDOSO

A DECENTRALISED ONLINE MULTI-AGENT PLANNING FRAMEWORK FOR MULTI-AGENT SYSTEMS

Porto Alegre

2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

A DECENTRALISED ONLINE MULTI-AGENT PLANNING FRAMEWORK FOR MULTI-AGENT SYSTEMS

RAFAEL CAUÊ CARDOSO

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Supervisor: Prof. Dr. Rafael Heitor Bordini

Porto Alegre
2018

Ficha Catalográfica

C268d Cardoso, Rafael Cauê

A decentralised online multi-agent planning framework for multi-agent systems / Rafael Cauê Cardoso . – 2018.
156 p.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rafael Heitor Bordini.

1. multi-agent planning. 2. multi-agent systems. 3. hierarchical task network. 4. goal allocation. 5. online planning. I. Bordini, Rafael Heitor. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Rafael Cauê Cardoso

A Decentralised Online Multi-Agent Planning Framework for Multi-Agent Systems

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on March 27, 2018.

COMMITTEE MEMBERS:

Prof. Dr. Luís Alvaro de Lima Silva (UFSM)

Prof. Dr. Sebastian Sardina (RMIT)

Prof. Dr. Luiz Gustavo Leão Fernandes (PPGCC/PUCRS)

Prof. Dr. Rafael Heitor Bordini (PPGCC/PUCRS - Supervisor)

A DECENTRALISED ONLINE MULTI-AGENT PLANNING FRAMEWORK FOR MULTI-AGENT SYSTEMS

ABSTRACT

Multi-agent systems often contain dynamic and complex environments where agents' course of action (plans) can fail at any moment during execution of the system. Furthermore, new goals can emerge for which there are no known plan available in any of the agents' plan library. Automated planning techniques are well suited to tackle both of these issues. Extensive research has been done in centralised planning for single-agents, however, so far multi-agent planning has not been fully explored in practice. Multi-agent platforms typically provide various mechanisms for runtime coordination, which are often required in online planning (i.e., planning during runtime). In this context, decentralised multi-agent planning can be efficient as well as effective, especially in loosely-coupled domains, besides also ensuring important properties in agent systems such as privacy and autonomy. We address this issue by putting forward an approach to online multi-agent planning that combines goal allocation, individual Hierarchical Task Network (HTN) planning, and coordination during runtime in order to support the achievement of social goals in multi-agent systems. In particular, we present a planning and execution framework called Decentralised Online Multi-Agent Planning (DOMAP). Experiments with three loosely-coupled planning domains show that DOMAP outperforms four other state-of-the-art multi-agent planners with regards to both planning and execution time, particularly in the most difficult problems.

Keywords: multi-agent planning, multi-agent systems, hierarchical task network, goal allocation, online planning.

UM FRAMEWORK DE PLANEJAMENTO MULTIAGENTE ONLINE E DESCENTRALIZADO PARA SISTEMAS MULTIAGENTES

RESUMO

Sistemas multiagentes freqüentemente contêm ambientes complexos e dinâmicos, nos quais os planos dos agentes podem falhar a qualquer momento durante a execução do sistema. Além disso, novos objetivos podem aparecer para os quais não existem nenhum plano disponível. Técnicas de planejamento são bem adequadas para lidar com esses problemas. Há uma quantidade extensa de pesquisa em planejamento centralizado para um único agente, porém, até então planejamento multiagente não foi completamente explorado na prática. Plataformas multiagentes tipicamente proporcionam diversos mecanismos para coordenação em tempo de execução, frequentemente necessários em planejamento online. Neste contexto, planejamento multiagente descentralizado pode ser eficiente e eficaz, especialmente em domínios fracamente acoplados, além de garantir algumas propriedades importantes em sistemas de agentes como privacidade e autonomia. Nós abordamos esse problema ao apresentar uma técnica para planejamento multiagente online que combina alocação de objetivos, planejamento individual utilizando rede de tarefas hierárquicas (HTN), e coordenação em tempo de execução para apoiar a realização de objetivos sociais em sistemas multiagentes. Especificamente, nós apresentamos um framework chamado Decentralised Online Multi-Agent Planning (DOMAP). Experimentos com três domínios fracamente acoplados demonstram que DOMAP supera quatro planejadores multiagente do estado da arte com respeito a tempo de planejamento e tempo de execução, particularmente nos problemas mais difíceis.

Palavras-Chave: planejamento multiagente, sistemas multiagentes, rede de tarefas hierárquicas, alocação de objetivos, planejamento online.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

A&A – Agents and Artifacts

AI – Artificial Intelligence

BDI – Belief-Desire-Intention

BNF – Backus–Naur Form

CDM – Centre for Disaster Management

CODMAP-15 – 2015 Competition of Distributed and Multi-Agent Planners

DMAP – Distributed and Multi-Agent Planning

DOMAP – Decentralised Online Multi-Agent Planning framework

ICAPS – International Conference on Automated Planning and Scheduling

IPC – International Planning Competition

HTN – Hierarchical Task Network

MA-HTN – Multi-Agent Hierarchical Task Network

MAP – Multi-Agent Planning

MAPC – Multi-Agent Programming Contest

MAS – Multi-Agent Systems

MDP – Markov Decision Process

PDDL – Planning Domain Definition Language

PRS – Procedural Reasoning System

UAV – Unmanned Aerial Vehicle

UGV – Unmanned Ground Vehicle

USV – Unmanned Surface Vehicle

XML – eXtensible Markup Language

CONTENTS

1. INTRODUCTION

Intelligent Agents are rational agents that perceive the environment and act upon it in pursuit of their goals. Multi-Agent Systems (MAS) are composed of multiple interacting agents that work to solve problems that are beyond the capabilities or knowledge of each individual agent [48]. MAS are often situated in dynamic, unpredictable environments where new plans of action may need to be devised if the overall system's goals are to be achieved successfully. For example, agents that join an organisation can pursue social plans (i.e., groups of structured social/global goals), which may fail at any point, prompting the creation of new social plans. Automated Planning is the act of formulating a plan of action in order to achieve some goal. Therefore, employing planning techniques at runtime of a MAS can improve agents' plans using knowledge that was not previously available before execution, or allow them to create new plans to achieve some goal for which there was no known effective course of action at design time, or even replan for social goals that had their social plans fail during execution.

Using agents as first-class abstractions in planning can be characterised, according to [49], into: single-agent planning for a single agent, single-agent planning for multiple agents, multi-agent planning for a single agent, and multi-agent planning for multiple agents. The latter is what we refer to when we use the term Multi-Agent Planning (MAP), that is, planning is done *by* multiple agents, and results in plans *for* multiple agents.

Research on Automated Planning has been largely focused on single-agent planning over the years. In centralised single-agent planning, agents are not considered as first-class abstractions, they are instead treated just like any other object described in the planning formalism. On the other hand, MAP has received increasing attention recently [50, 131, 93], tackling new and complex multi-agent problems that require decentralised solutions. One such problem is to combine planning and execution [91]. By allowing agents to do their own individual planning, the search space is effectively pruned, which can potentially decrease planning time on loosely-coupled domains. Agents also get to keep some privacy within the system, and maintain their autonomy. However, for individual planning to be feasible, goals have to be be efficiently preallocated, and agents have to coordinate before or after planning, or during runtime.

MAS development platforms also changed the focus from agent- to organisation-centred approaches. Recent research, as evidenced in [13, 118], shows that considering other programming dimensions such as environments and organisations as first-class abstractions (along with agents) allow developers to create more complex MAS. Multi-agent programming languages and platforms that cover the social and environmental dimensions of multi-agent systems, as well as the agent dimension [14], are what make

multi-agent oriented programming especially suited for solving complex problems that require highly social, autonomous software.

To demonstrate an application of some of these concepts of MAP and MAS, as applied in this thesis, consider this setting. There is a group of multiple robots that are working together in search and rescue operations after a natural disaster, wherein high-level control of each robot is done by a different agent directly integrated into the robot's hardware. These robots are heterogeneous, with each robot containing its own set of private knowledge, goals, and plans. These robots are all part of the same organisation, thus sharing the same social goals that they cooperate to fulfil. Search and rescue environments are dynamic, constantly changing, which requires agents to be able to find solutions at runtime.

Our main contributions in this thesis are the design, implementation, and evaluation of the Decentralised Online Multi-Agent Planning framework (DOMAP). DOMAP is divided into several main components: i) multi-agent system factored representation, a multi-agent planning formalism that contains information about the world according to each agent's point of view; ii) a contract net protocol mechanism for goal allocation; iii) individual Hierarchical Task Network (HTN) planning; iv) social laws to coordinate agents at runtime. Solutions found by DOMAP are generally sub-optimal, since planning at runtime often requires fast response, although finding optimal solutions are possible and usually reliant on an optimal goal allocation. Although approaches to online single-agent planning usually involve some kind of interleaving planning and execution (e.g., lookahead planning), in our approach to online multi-agent planning we focus on domains that allow agents some time to plan while the system is still in execution.

We implemented DOMAP in JaCaMo [13], a platform for the development of MAS with the environment, Belief-Desire-Intention (BDI) agents, and organisation as first-class programming abstractions. Our approach combines MAP with MAS, allowing for dynamic execution of solutions found through planning, and making it easier to transition from planning to execution and vice-versa. We use three domains in our experiments, the classical Rovers domain, our novel Floods domain, and the well established Petrobras domain [135]. Our experiments have fully cooperative and norm-compliant agents that aim to achieve their organisations' social plans. To evaluate our implementation, we selected four state-of-the-art planners that took part in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP-15) [138]. DOMAP outperforms those other planners in the largest planning problems, and is the best overall when considering planning and execution together.

1.1 Motivation

Although it is possible to adapt centralised single-agent techniques to work in a decentralised way, such as in [36], decentralised computation is not the only advantage of using MAP. By allowing agents to do their own individual planning the search space is effectively pruned, which can potentially decrease planning time on domains that are naturally distributed. This natural decentralisation also means that agents get to keep some (or even full) privacy from other agents in the system, as they might have beliefs, goals, and plans that they do not want to share with other agents.

Combining automated planning with autonomous agents results in online (continual) planning, where agents can consecutively plan and execute during the system's runtime. Multiple approaches to single-agent planning in this context (i.e., centralised planning generating plans for single agents) can be found in a recent survey [80]. Our approach however, combines MAP with MAS, resulting in a decentralised online multi-agent planning framework for multi-agent systems.

Much work has already been done to add planning into autonomous agents. Formal semantics for adding HTN planning in the BDI agent programming language CAN generated the CANPLAN language, described in [109] and further extended in [40, 111, 41, 39], which provides operational semantics and constructs for HTN planning that can be used in BDI-based agent programming languages. IndiGolog [57] is an extension of Golog [76], an agent-oriented programming language based on the situation calculus and implemented in Prolog, that adds planning in the form of high-level program execution, allowing planning to be incrementally realised by interleaving planning and execution. The GOAL [60] agent oriented programming language was also extended to include planning in [61], where it is shown that GOAL can also act as a planning formalism and can solve a subset of PDDL (Planning Domain Definition Language) problems.

There is also some older work found in RETSINA [123], TAEMS [45], and Machinetta [112]. RETSINA has no goal allocation mechanism, they plan for pre-allocated goals; their approach to coordination problems is to monitor and replan, while we use social laws with the possibility of replanning. TAEMS and Machinetta both have no explicit planning component, instead they focus on the use of scheduling techniques to coordinate tasks. These three approaches were discontinued and are no longer being developed, which limits their practical use (especially their use in new experiments).

Our work in this thesis differs from all of these other approaches in that we focus on multiple agents and their interaction in the agent, environment, and organisation dimensions of the MAS. We also provide a quantitative comparison of our implementation against other implemented multi-agent planners, something that is lacking in all of the aforementioned work.

1.2 Objectives

The main objectives addressed in this thesis are:

1. Design the Decentralised Online Multi-Agent Planning (DOMAP), a framework for formulating plans of action for social goals at runtime of a MAS;
2. Implement the framework by integrating it with a MAS development platform (JaCaMo);
3. Evaluate our framework against other state-of-the-art and recently developed multi-agent planners;

1.3 Thesis Outline

This thesis is structured as follows. In the next chapter, some background on Automated Planning and Intelligent Agents is presented, with an emphasis on the topics relevant to better understand the contributions of our work, as well as a discussion on related work, explaining some of the key differences between other approaches and DOMAP. In Chapter 3, introduces our Floods domain, a novel domain designed specifically for online multi-agent planning. Chapter 4 contains the design of the Decentralised Online Multi-Agent Planning framework (DOMAP), followed by detailed information about each of the major components in the framework: input language, goal allocation, individual planning, and coordination mechanism. Chapter 5 provides DOMAP implementation details into how we implemented it in the JaCaMo MAS development platform. In Chapter 6 we describe our experiments' settings, show results of the evaluation of DOMAP against four other state-of-the-art multi-agent planners, and then discuss some of the more important results. This thesis concludes in Chapter 7, with a summary of our contributions and discussion of future work.

2. BACKGROUND AND RELATED WORK

In this chapter, we cover the background on some fundamental topics related to our approach. We start by describing automated planning and its applications, followed by a more detailed view on HTN planning, HTN planning in agent-oriented programming languages, and on multi-agent planning. Next, basic concepts of intelligent agents are introduced, followed by some additional information on MAS, multi-agent programming, and JaCaMo (the MAS development platform we used to implement DOMAP). Finally, the most relevant related work is discussed.

2.1 Automated Planning

Automated planning is the computational study of planning, which is an abstract deliberation process on choosing and ordering actions in order to achieve goals [89]. This is done by anticipating the outcome of these actions, which in turn can cause the deliberation process to take some time to find the best possible solution. Thus, in domains that require fast planning times, returning a sub-optimal solution can be the best approach.

Planning can be applied to a myriad of problems, and as such, there are many forms of planning available: path and motion planning, perception planning, communication planning, task planning, and several others [89]. Even though most planning techniques can be applied in many of these forms, in the framework presented in this thesis we focus on task planning. This does not mean movement cannot be considered, as long as it can be represented symbolically, for example, if a robot has the goal of moving from place A to place C and the solution found is to move from A to B and from B to C. then this is a movement task. How the robot moves from A to B is left for the underlying system (e.g., a path-finding algorithm).

There are two approaches that planners can take regarding domains [89], domain-independent and domain-specific planning. Domain-specific usually outperforms domain-independent planning, but it is restricted to that specific domain and will not work if there are any deviations from it. Our framework, DOMAP, is domain-independent and can be used in a variety of loosely-coupled domains, though the use of domain-specific knowledge can be used to improve its performance on more tightly-coupled domains via social laws.

In Figure 2.1 [89], we show a simple conceptual model for a traditional planner. Σ represents the *state-transition system*, it is governed by a state-transition function according to the events and actions that it receives. The *controller* outputs an action according to some plan and the related state of the system (observations). A *planner*

uses the description of the system Σ, the initial state, and the goals (objectives) to formulate a plan for the controller in order to achieve its goals. It is worth noting that offline planners do not include a controller (i.e., the execution component) and thus the implementation and execution of the plans that were found is up to the developer. In our approach the planner is inherently integrated with the controller, thus allowing it to find and execute plans.

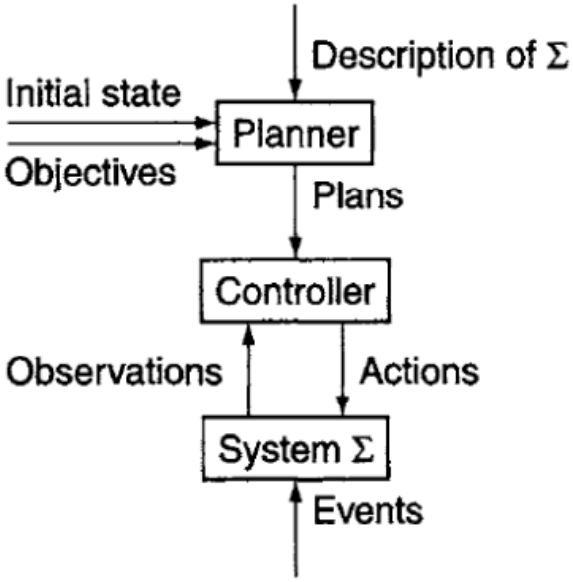

Figure 2.1 – Traditional architecture for offline planning [89].

Even though the computational cost of automated planning is still high for complex and dynamic domains, there has been several success stories of real-world applications. As an example there is the control of spacecraft Deep Space I, that successfully completed its goal by encountering the Comet Borrelly and capturing several images and other scientific data. The control was done by the Autonomous Remote Agent, based on automated planning techniques. The Hubble Space Telescope also uses planning techniques [83], the SPSS is its short-term planning system and Spike its long-term.

To deal with these various forms of planning and different application needs, many types of planning techniques emerged through the years. The most popular is Classical Planning, also known as STRIPS-like planning, which was an early planner for restricted state-transition systems [53]. STRIPS action theory is still used in many recent planners, and is the foundation of most classical planners, including the well known formalism for representing planning problems, PDDL [79]. There are many other types that aim to discard some of the assumptions made in classical planning, such as in temporal planning, where temporally overlapping actions are possible, and probabilistic planning, allowing partial observability of the environment.

Regarding search algorithms for planning, there are mainly two different approaches: state-space and plan-space. In state-space search [90], each node corresponds

to a state of the world, each arc corresponds to a state transition, and the plan corresponds to the path taken in the search space. The previously discussed STRIPS planner uses this type of search, as well as most classical planners. In a plan-space search [90], nodes are partially specified plans. Arcs are plan refinement operations intended to further complete a partial plan, i.e., to achieve an open goal or to remove a possible inconsistency. A refinement operation avoids adding to the partial plan any constraint that is not strictly needed for addressing the refinement (least commitment principle). Planning starts from an initial node corresponding to an empty plan, until arriving in a final node containing a solution to achieve the required goals.

The difference between plan-space and state-space is not only in its search space, but also in its definition of a solution. Plan-space search algorithms use a more intricate plan structure than just a sequence of actions as in search-space. The solution of a plan-space search is comprised of the choice of actions, causal links, and partial ordering of those actions.

2.1.1 Classical Planning

Classical STRIPS planning consists in sequences of actions that transition the world from an initial state to a state satisfying a goal condition. States are modelled as sets of propositions that are true in those states, and actions can change validity of certain propositions.

A formal definition of classical planning, from [7], is as follows: let P be a set of all propositions modelling properties of world states. Then a state $S \subseteq P$ is a set of propositions that are true in that state, while every other proposition is false. Each action a is described by four sets of propositions $(B_a^+, B_a^-, A_a^+, A_a^-)$, where $B_a^+, B_a^-, A_a^+, A_a^- \subseteq P, B_a^+ \cap B_a^- = \emptyset, A_a^+ \cap A_a^- = \emptyset$. Sets B_a^+ and B_a^- describe positive and negative preconditions (before constraints) of action a, that is, propositions that must be true and false right before the action a. Action a is applicable to state S iff $B_a^+ \subseteq S \wedge B_a^- \cap S = \emptyset$. Sets $A_a^+$and A_a^- describe positive and negative effects (after constraints) of action a, that is, propositions that will become true and false in the state right after executing the action a. If an action a is applicable to state S then the state right after the action a is

$$\gamma(S, a) = (S \setminus A_a^-) \cup A_a^+. \tag{2.1}$$

If an action a is not applicable to state S then the state transition $\gamma(S, a)$ is undefined.

A classical planning problem consists of a set of actions A, a set of propositions S_0 called an initial state, and disjoint sets of goal propositions G^+ and G^- that are required to be true and false in the goal state. A solution to the planning prob-

lem is a sequence of actions $a_1, a_2, \ldots, a_n$ such that $S = \gamma(...\gamma(\gamma(S_0, a_1), a_2), ..., a_n)$ and $G^+ \subseteq S \wedge G^- \cap S = \emptyset$. This sequence of actions is called a *plan*.

2.1.2 HTN Planning

Similarly to classical planning, in HTN planning [89, Chapter 11] each state of the world is represented by a set of atoms and each action results in a deterministic state transition. The difference is that besides the operators (action description) present in both approaches, HTN planning also includes a set of *methods*. Methods are recipes on how to decompose tasks into smaller subtasks. These methods are applied to the initial task network (goals) until a primitive task is reached, that is, some planning operator can be applied. This convenient way of writing recipes is more closely related to how a human expert would think about solving a problem, thus making HTN planning more suited for practical applications. Furthermore, the extra domain information contained in methods usually results in better performance when compared to typical classical planners.

Domains in HTN planning contain a set of operators and a set of methods. Operators are action descriptors that can be executed given some preconditions, causing a list of postconditions to become true. They can cause a state transition to occur in the system, while methods can only decompose tasks into smaller subtasks, which can eventually lead to primitive tasks. Methods are non-primitive tasks that impose constraints into the domain, in order to guide the search for solutions by effectively pruning some states. An HTN problem description contains a list of atoms that are true during the initial state of the system, as well as the initial task network.

There are two cases where a plan π can be the solution for a problem P, as described in [89]. It depends if the initial task network is primitive or non-primitive. If it is primitive, then a plan π with actions $(a_1, a_2, ..., a_k)$ is a solution for P if there is a ground instance w' of the task network w and a total ordering of the task nodes where the plan π is executable in state s_0, and every constraint between the task nodes of the task network holds. Otherwise, if the initial task network is non-primitive, then π is a solution for P if a sequence of task decompositions can be applied to w in order to produce a primitive task network w' such that π is a solution for w'.

Algorithm 2.1 [89] contains the main function of an abstract HTN planner. The parameters are: s state-transition system, U set of task nodes, C set of constraints, O set of operators, and M set of methods. Line 4 checks if task nodes are primitive, and if so they are decomposed. Otherwise, in line 11, if task nodes are non-primitive then a viable method that was not previously applied is selected and calls the function again with the

new subtasks and relevant constraints generated. Every task in U must eventually be decomposed. The *mgu* reference on line 15 is a simple unification function.

Algorithm 2.1: An abstract algorithm for HTN planning [89].

```
1  Function Abstract-HTN (s, U, C, O, M)
2      if (U,C) can be shown to have no solution then
3          return failure;
4      else if U is primitive then
5          if (U,C) has a solution then
6              non-deterministically let π be any such solution;
7              return π;
8          end
9          else
10             return failure;
11         end
12     else
13         choose a non-primitive task node u ∈ U;
14         active ← {m ∈ M | task(m) is unifiable with t_u};
15         if active ≠ ∅ then
16             non-deterministically choose any m ∈ active;
17             σ ← call mgu for m and t_u that renames all variables of m;
18             (U',C') ← δ(σ(U,C),σ(u),σ(m));
19             return Abstract-HTN(s, U', C', O, M);
20         end
21         else
22             return failure;
23         end
24     end
25 end
```

The SHOP2 planner[1] [90] is a well-known implementation of HTN planning. SHOP2 is written using the Lisp high-level programming language. Although there are versions of it available in Java (JSHOP2) and in Python (Pyhop), SHOP2 contains more features than any other version. Some of these features include different search strategies: depth-first stopping at the first plan found (the only one available in all versions); depth-first search for the shallowest (lowest-cost) plan, and iterative-deepening search. Another important feature present in SHOP2 is the *time-limit* argument, which allows us to set a time-limit for the search, and can be especially useful in online planning, where the system might require fast action responses. This time-limit argument is inspired by the notion of anytime planning as defined by [44]. SHOP2 also has internal operators (denoted by double exclamation points) that are used internally in the planning process, but do not correspond to any action in the solution, such as book keeping operators to mark places visited in a path.

To illustrate some of the basic concepts of HTN planning, consider the example in Listing 2.1. The domain is defined on lines 1 through 10. An operator has a precon-

[1] https://www.cs.umd.edu/projects/shop/

dition list, a delete list, and an add list. In this domain there are two operators, the first is *!pickup ?a*, it has an empty precondition and delete list, and an add list where it adds the fact *have ?a*, with *?a* being a variable that is unified by the planner. The second operator, *!drop ?a*, contains a precondition and a delete list, but no add list. If the precondition *have ?a* holds, then the fact *have ?a* is removed, i.e., the item is dropped. A method has a precondition list and a task list (containing substasks such as primitive and non-primitive tasks). The method *swap ?x ?y* in this example is branched into two, in the first case if the precondition *have ?x* does not hold then the second branch is activated, where the precondition is *have ?y*. In other words, if it has the first object it will drop and pickup the second, otherwise it drops the second and pickup the first. The problem is defined on lines 11 through 13, and contains the fact *have guitar* and the initial task network (goal) *swap guitar violin*. Both are used by the planner to unify with method's and operator's variables during planning.

Listing 2.1 – Basic example of SHOP2 planning formalism.

```
1  (defdomain basic-example (
2    (:operator (!pickup ?a) () () ((have ?a)))
3    (:operator (!drop ?a) ((have ?a)) ((have ?a)) ())
4    (:method (swap ?x ?y)
5      ((have ?x))
6      ((!drop ?x) (!pickup ?y))
7      ((have ?y))
8      ((!drop ?y) (!pickup ?x))
9    )
10 ))
11 (defproblem problem1 basic-example
12   ((have guitar)) ((swap guitar violin))
13 )
```

Figure 2.2 shows the task decomposition dony by the SHOP2 planner for the basic example provided in Listing 2.1. The initial task network is decomposed into a non-primitive subtask where the variables are instantiated and checked against the precondition of the *swap* method. Because the precondition holds, the task list of the method is further decomposed into two new primitive subtasks, that when executed in order (first drop and then pickup), achieve the goal task. The *swap* method has two possible branches, and because the precondition from the first branch holds, SHOP2 does not need to expand the second branch unless it exhausted the first branch and found no solution. Depending on the search method used in SHOP2, it can also keep trying to search for a plan with least cost. Either way, the planner would then backtrack and try to expand the second branch.

The various versions of SHOP have been widely used in many different applications [88]. Some government projects include: the US Naval Research Laboratory's Hierarchical Interactive Case-Based Architecture for Planning (HICAP) - a system for helping experienced human planners develop evacuation plans; and the Naval Research

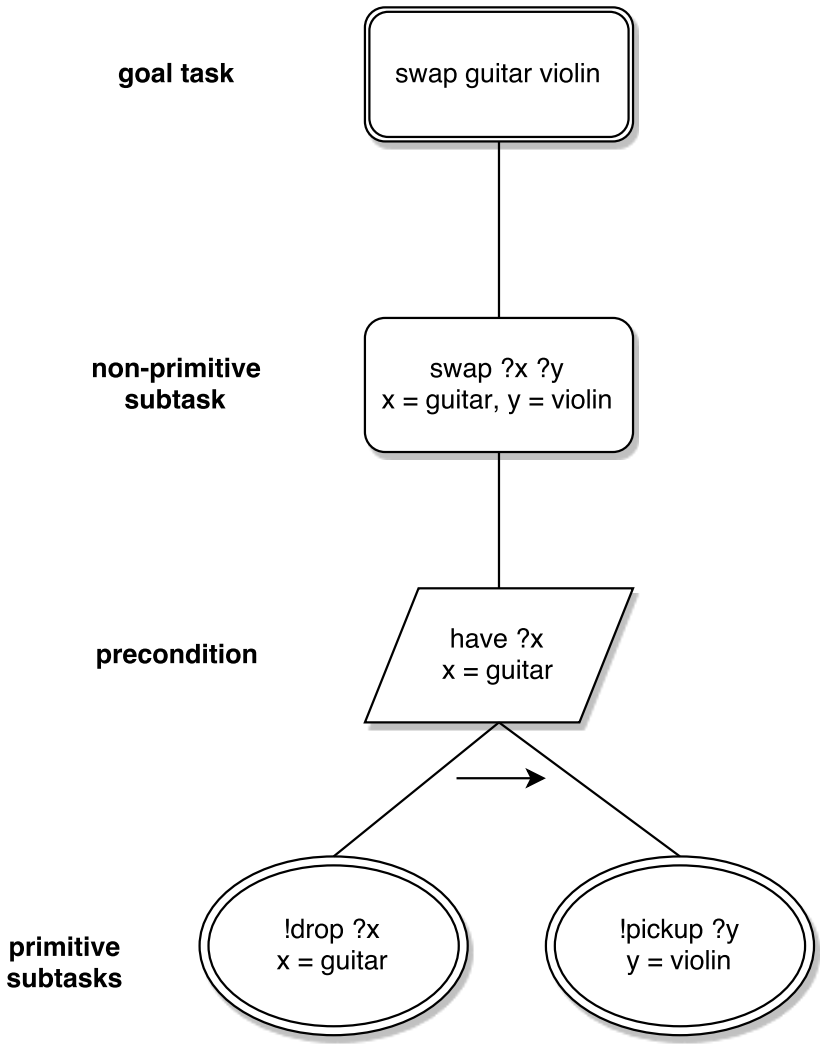

Figure 2.2 – Step-by-step solution to SHOP2 basic example.

Laboratory's Analogical Hypothesis Elaboration for Activity Detection (AHEAD) - a system for evaluating terrorist threats. As for industry projects: the Smart Information Flow Technologies company uses SHOP2 to control multiple Unmanned Aerial Vehicles (UAVs); and the Infocraft company uses SHOP2 in their system for materials selection in a manufacturing process.

HTN Planning in Agent-Oriented Programming Languages

In dynamic systems it is not feasible for a planner to know all details of the dynamics in the system, but it cannot completely ignore how they can evolve overtime. It needs to check online if a solution remains valid, and if it does not, then revise or replan. Figure 2.3 [89] expands on Figure 2.1 by adding the feedback of the plan's execution status from the controller to the planner. This allows for a more realistic model, with plan supervision, plan revision, and re-planning mechanisms [89].

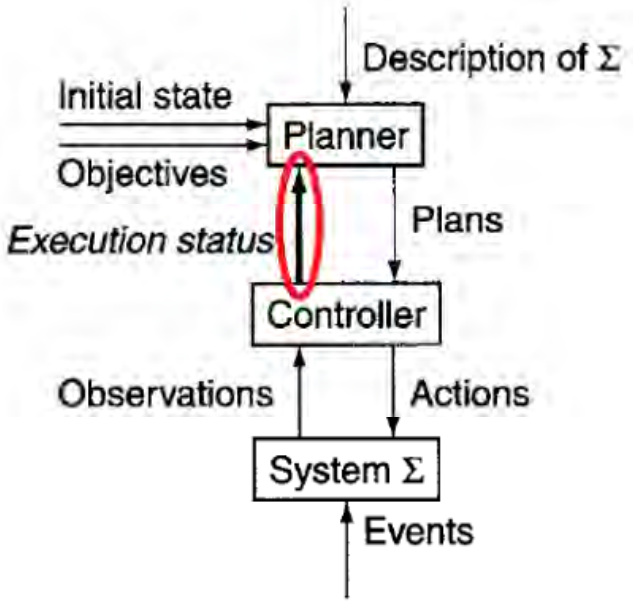

Figure 2.3 – Conceptual model for online planning [89].

Online planners are much harder to benchmark then offline planners, as performance no longer depends only on the planner, but on the system as a whole, including the controller. Many domain-independent planning algorithms have been integrated into agent programming languages, but most real world applications make use of online planners that are domain-specific, as they can achieve the best performance. A recent survey [80] shows some of these integrations between planning algorithms and BDI (Belief-Desire-Intention) agents. These, however, are single-agent approaches that do not consider the decentralised nature of MAP and MAS.

An HTN-method, as defined in [39], $m = [\![\tau(t), [S, \phi_{htn}]]\!] \in Me$ — where $\tau(t)$ is either a primitive or non-primitive task, S is a task list, ϕ_{htn} is the set of constraints, and Me is the set of methods — can be transformed into a BDI plan $\rho_m = e_m : \psi_m \rightarrow P_m$, where the event goal $e_m = \tau(t)$, the context condition $\psi_m = \{l | (l, n_1)$ occurs in $\phi_{htn}\} \cup \{\neg l | \neg(l, n_1)$ occurs in $\phi_{htn}\}$, and the plan body $P = (P_1 || \ldots || P_m)$. The event goal using the corresponding HTN non-primitive task terms t. The context condition contains the conjunction of literals that need to hold before the first task from the task list. Finally, the plan body consists of tasks translations from the task list into subgoals/subplans if the task is non-primitive, or actions if the task is primitive. In our approach, however, we do the opposite, transforming a BDI plan ρ into an HTN method m_ρ. A summary of these translations from BDI to HTN entities is shown in Table 2.1 [109].

The work done with the CANPLAN language [40, 111, 41, 39] has laid much of the formal work for using HTN planning in BDI agents. CANPLAN2 [110], a modular extension of CANPLAN [109], is a BDI-based formal language that incorporates an HTN planning mechanism. This approach was further extended in [111] to address previous limitations, adding features such as failure handling, declarative goals, and lookahead

Table 2.1 – Mappings from BDI to HTN entities [109].

BDI entities	**HTN entities**
belief base B	state I
action	primitive task
belief operations $+b$ and $-b$	primitive tasks
event goal $!e$	compound task
test goal $?\phi$	state constraints
plan context condition ψ	state constraints
plans in sequence $P; P'$	ordering constraints
plans in parallel $P \| \| P'$	no ordering constraints
plan body P	task network $d = [S, \phi]$
plan $\rho = e : \psi \rightarrow P$	$m = [\![\tau, d]\!]$
plan library Π	set of methods Me

planning. The CAN programming languages are not implemented, although its formalism could be used to augment existing BDI-based agent languages. Similarly, in [40] they proposed an approach to obtain new abstract plans in BDI systems for hybrid planning problems, where both goal states and the high-level plans already programmed are considered, bringing classical planning into BDI hierarchical structures.

2.1.3 Multi-Agent Planning

Over the years, Multi-Agent Planning (MAP) has been interpreted as many different things, but they can be characterised into two main concepts. Either the planning process is centralised and produces distributed plans that can be acted upon by multiple agents, or the planning process itself is multi-agent. Recently, the planning community has been favouring the concept that MAP is actually both of these things, that is, the planning process is done *by* multiple agents, and the solution's plans are *for* multiple agents.

Planning with/for agents can be characterised into [49]: single-agent planning for a single agent — a centralised planner searches for a centralised solution; single-agent planning for multiple agents — a centralised planner searches for a solution and then distributes it among agents; multi-agent planning for a single agent — agents cooperate to form a centralised plan; and multi-agent planning for multiple agents — agents cooperate to form individual plans, dynamically coordinating their activities along the way. The latter is what we refer to when we use the term Multi-Agent Planning (MAP), that is, planning is done *by* multiple agents, and results in distributed plans *for* multiple agents.

Durfee also establishes some stages of multi-agent planning in [49], further extended in [43]:

1. **Global goal refinement**: decomposition of the global goal into subgoals;
2. **Task allocation**: use of task-sharing protocols to allocate tasks (goals and subgoals);
3. **Coordination before planning**: coordination mechanisms that avoid conflicts before planning;
4. **Individual planning**: planning algorithms that search solutions for the problem;
5. **Coordination after planning**: coordination mechanisms that solve conflicts after planning;
6. **Plan execution**: agents carry out plans from the solution found.

Not all of these phases are mandatory, and some may even be combined into one. Our DOMAP framework, includes phases 2, 3, and combines 5 and 6.

In Table 2.2 we characterise and summarise the differences between single-agent planning and multi-agent planning. While multi-agent planning could have no privacy, even in fully cooperative domains it is fairly trivial to allow at least some sort of partial privacy. Full privacy, on the other hand, is quite difficult because of the coordination usually required in some multi-agent planning problems. Single-agent planning can have no privacy, since the planner needs all the information available. Agent abstractions in single-agent planning are usually represented as any other object or fact in the environment. In multi-agent planning agents are treated as first-class abstractions, where each agent can have its own domain and problem specification.

Table 2.2 – Comparisons between single-agent planning and multi-agent planning.

	computation	**privacy**	**agent abstraction**
single-agent planning for a single agent	centralised	not needed	not needed
single-agent planning for multiple agents	centralised	none	objects
multi-agent planning for a single agent	decentralised	none or partial	not needed
multi-agent planning for multiple agents	decentralised	partial or full	first-class

When considering multiple agents, planning can become increasingly more complex, giving rise to several problems [50]. Actions that agents choose to execute may cause an impact in future actions that other agents could take. Likewise, if an agent knows what actions other agents plan to take, it could change its own current choices. When dealing with multiple agents, concurrent actions are also a possibility that can

cause major impact in performance and consequently in planning for optimal or suboptimal solutions. These are some of the problems that drive the research on MAP.

MAP can be used in various application areas [89] such as multi-robots environments, cooperating software distributed over the Internet, logistics, manufacturing, disaster management, evacuation operations, and games.

2.2 Intelligent Agents

Russel and Norvig state [106] that "An agent is anything that can be viewed as perceiving its environment through *sensors* and acting upon that environment through *actuators*". Therefore, sensor data, i.e. perceptions, are the input of an agent and the actions its output [144]. This concept, illustrated in Figure 2.4 [106], resembles a similar concept of robots, which are also known to be equipped with sensors and actuators. The question mark represents the reasoning that the agent performs using perceptions captured by the sensors, and the appropriate action output that is executed by its actuators.

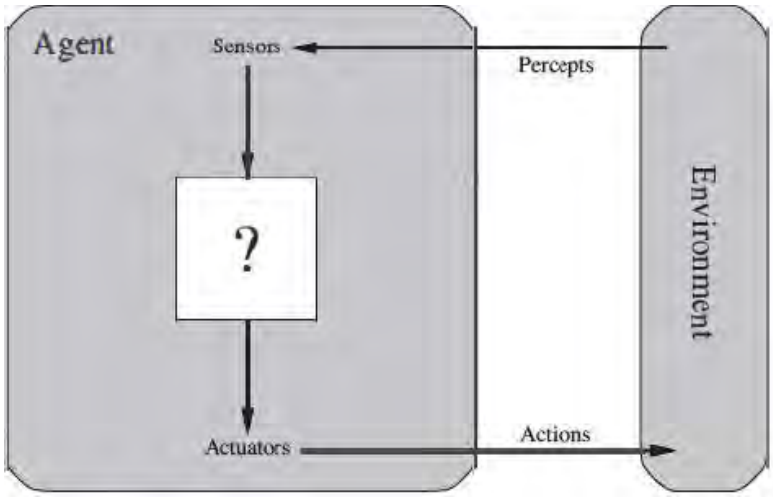

Figure 2.4 – A generic agent architecture [106].

The reasoning mechanism represented by the question mark in Figure 2.4 can be, for example, the Procedural Reasoning System (PRS) [56]. In this system an agent has a library of pre-compiled plans that are composed of:

- **goal**: postcondition of the plan;
- **context**: precondition of the plan;
- **body**: the course of action to carry out.

Another important feature of PRS is the intention stack. It contains all goals that have not yet been achieved, and it is used by the agent to search its library for plans

that match the goal on the top of the stack. If the precondition of a plan is satisfied, then it becomes a possible option to be executed by the agent.

The PRS was largely based in the BDI architecture. First described in [19] and further extended by [101], the BDI architecture tries to model the process of deciding which action to perform to achieve certain goals. It has three primary mental attitudes: *belief* — what the agent believes to be true about its environment and other agents; *desire* — the desired states that the agent hopes to achieve; and *intention* — a sequence of actions that an agent wants to carry out in order to achieve a goal. These mental attitudes respectively represent the information, motivational, and deliberative states of the agent. Figure 2.5 [145] illustrates the workflow in a generic BDI architecture: the *belief revision* function receives input information from the sensors, and it is responsible for updating the belief base. This update can generate more options that can become current desires based on the belief base and the intentions base. The *filter* is responsible for updating the intentions base, taking into account its previous state and the current belief base and desire base. Finally, an intention is chosen to be carried out as an action by the agent.

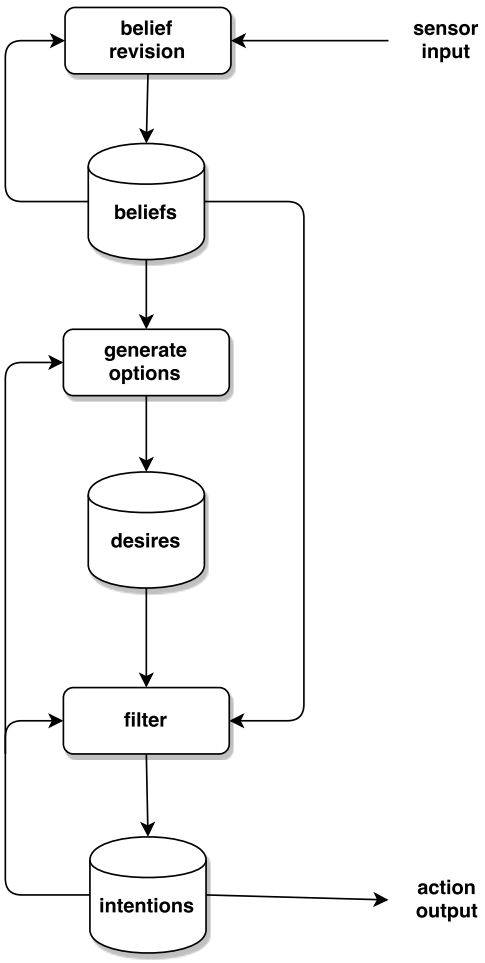

Figure 2.5 – Generic BDI model, adapted from [145].

According to Wooldridge in [145], an agent has the following characteristics:

- **pro-activeness**: goal-directed behaviour, agents take the initiative;
- **reactivity**: agents react to changes in their environment as they happen;
- **social ability**: agents are capable to communicate and interact with other agents.

These three characteristics complement nicely the requirements in online multi-agent planning. Pro-activeness is necessary by agents to decide when is the best moment to start planning. Reactivity is needed to adapt to changes in the environment while planning is underway. Social ability is vital for coordination in order to avoid any conflicts that may arise during planning.

Systems that require the use of intelligent agents will seldom need only a single agent. In the next section we briefly go through MAS and some of their applications.

2.2.1 Multi-Agent Systems

Multi-Agent Systems (MAS), as defined in [144], "are systems composed of multiple interacting computing elements, known as agents". These agents usually take part in an organisation or a society, interacting together and often cooperating to achieve some organisational goal [51].

Organisations in a multi-agent system are complex entities in which agents interact in order to achieve some global purpose [48]. They provide scope for these interactions, reduce or manage uncertainty, and coordinate agents to improve efficiency. They are especially relevant to MAS in complex, dynamic, and decentralised environments. These environments are very similar to domains found in multi-agent planning.

For example, if we think of an university as a MAS organisation, it has its own goals (e.g., searn prestige and/or money), plans (e.g., admit students, teach classes, make research), and group roles (e.g., deans, professors, students, staff). But in order to achieve its goals, it needs individuals (agents) to fulfil group roles that can then pursue the university (organisation) goals [48]. Agents may also have private goals that can only be achieved through the organisation, e.g., a student pursuing a degree in computer science needs to join an university.

Environments also play a fundamental part in agent-based systems. They can be virtual or physical, or both in some cases, as it can be beneficial to simulate parts of a physical environment as virtual elements. There are two main different views on the concept of environments in MAS. In classical Artificial Intelligence (AI), environments represent the external world that is perceived and acted upon by agents in order to achieve their goals [106]. A more recent view, shows the environment as a first-class

abstraction that encapsulates functionalities to support agent activities [142]. The latter shows the environment as not just being the target of agent actions and a generator of perceptions, but a part of the MAS just as important as any agent. Thus, this first-class abstraction of the environment in MAS can be considered as *endogenous*, while the classical AI environment is *exogenous* [103].

These *endogenous* environments can be considered to have three different levels of support, as identified in [142]:

i. **basic level**: provide support for resources that the MAS can interact with (e.g., sensors, actuators, printer, network);

ii. **abstraction level**: bridge the conceptual gap between agents and low level details regarding deployment of the environment, making them transparent to the programmer;

iii. **interaction-mediation level**: mediate the interaction between agents in the environment and regulate the access to shared resources.

This view allows the development of environments that can include some common MAS concepts, such as shared data objects, shared resources, and communication/-coordination services, all while keeping them domain-independent [103]. For example, the *stigmergic* coordination strategy in MAS cannot involve any communication. Considering only the agent abstraction it could be very difficult to develop such a strategy, but by including the environment abstraction it is possible to model, for example, a pheromone ground, such as the one described in [132].

Thus, multi-agent systems are composed of multiple interacting agents that work to solve problems that are beyond the capabilities or knowledge of each individual agent [48]. Sycara describes, in [122], MAS as systems with the following attributes:

- agents have incomplete knowledge or capabilities for solving the problem;
- there is no global control;
- information is decentralized;
- computation is asynchronous.

These are attributes that also classify distributed problem solvers, of which multi-agent planners are also a part of.

There are many examples [48] of possible applications of MAS. For instance, a logistics system where agents coordinate to transport and store a variety of goods, or a network of sensory agents that coordinate their activities to monitor traffic in busy intersections. In the next section we discuss some of the tools available to develop multi-agent systems. One of the first known uses [86] of robots in disaster management was

during the World Trade Center disaster, where several robots were deployed in urban search and rescue missions. More recently [87], several unmanned aerial vehicles were used in conjunction with social media to aid in the 2015 Memorial Day weekend floods in Texas, and to illustrate the need for decision-making support.

2.2.2 Multi-Agent Programming

Traditional software engineering and programming languages are generally not made with autonomy in mind, especially when there is a need for multiple software to work together to achieve the system's goals [14]. Shoham first introduced agent programming in 1993 [115], and over the years the agent community has been improving what were mainly theoretical approaches back then, into practical agent-oriented programming languages.

Many practical agent-oriented programming languages exist and continue to be developed and improved upon nowadays. Some examples include Jason [15], JACK [64], 2APL [38], GOAL [60], and the Agent Factory Framework [95]. More recently, there have been several new agent languages emerging, such as ALOO [102], ASTRA [33], and SARL [105].

Several studies indicate that Jason has better performance when compared with other agent-oriented programming languages. Jason was included in a qualitative comparison of features available in Erlang, Jason, and Java [67]; in a universal criteria catalogue for agent development artifacts [20]; in a quantitative analysis of 2APL, GOAL, and Jason regarding their similarity and the measured time it takes to reach specific agent states [9]; a performance evaluation of Jason when used for distributed crowd simulations [52]; an approach to query caching and a performance analysis of its usage in Jason, 2APL and GOAL [2]; an implementation of Jason in Erlang and a benchmark for evaluating its performance [47]; a quantitative comparison between Jason and two actor-oriented programming languages (Erlang and Scala) using a communication benchmark [26, 27]; and finally a performance evaluation of several benchmarks between agent programming languages (Jason, 2APL, and GOAL) and actor programming languages (Erlang, Akka, and ActorFoundry) [29]. In cases where performance was considered, Jason had excellent results. Jason controls the agent abstraction in the JaCaMo MAS development platform, which was one of the reasons for selecting JaCaMo as our agent platform implementation.

According to Bordini and Dix in [14], they state that:

> Originally agent programming languages were mostly concerned with programming individual agents, and very little was available in terms of programming abstractions covering the social and environmental dimensions of multi-agent systems as well as the agent dimension.

These multiple abstraction layers make multi-agent oriented programming especially suited for solving complex problems that require highly social, autonomous software.

Before describing some of the options available for the development of multi-agent oriented programming, we compiled some of the features that are expected to be present in these platforms, as reported in [14]:

- **Reacting to events** *vs* **long-term goals**: An agent needs to be able to appropriately react to events in the environment without compromising their long-term goals.
- **Courses of action depend on circumstances**: Ability to specify multiple courses of actions for a specific event, as it can often be triggered in many different circumstances.
- **Choosing courses of action only when about to act**: Environments in multi-agent applications are highly dynamic, thus, agents should only choose a particular course of action when they are about to act.
- **Dealing with plan failure**: Dynamic environments often result in many plan failures. Agents should be able to detect and deal with these failures.
- **Rational behaviour**: Agent applications require rationality. An agent should be able to reason on how to accomplish a goal, and to keep trying until it believes the goal to be accomplished, or otherwise impossible to accomplish at all.
- **Social ability**: Communication between agents is crucial for both cooperative and self-interested agents that need to share resources. As established in the previous section, organisations are very important in MAS.
- **Code modification at runtime**: Other agents, humans, or entities, should be able to send new plans to be added to an agent's plan library during execution. Thus, an agent's plan library should be able to be modified at runtime. This can also be allowed in organisations, so that norms can be created or modified on-the-fly.

Below we present some of the more robust MAS development platforms, containing multiple abstraction levels, found in the literature.

JADE[2] [10] is an open source platform for the development of peer-to-peer agent based applications. Besides the agent abstraction, it also provides: task execution and composition model, peer-to-peer agent communication based on asynchronous message passing, and a yellow page service that supports the publish and subscribe discovery

[2]http://jade.tilab.com/

mechanism. JADE-based systems can be distributed across machines with different operational systems, and has been used by many languages (e.g., Jason and JaCaMo) as a distribution infrastructure.

Jadex[3] [99] allows the programming of intelligent software agents in Java. The agent abstraction is based on the BDI model, and provides several features such as: a runtime infrastructure for agents, multiple interaction styles, simulation support, automatic overlay network formation, and an extensive runtime tool suite.

The TAEMS framework [45] provides a modelling language for describing task structures, i.e., tasks that agents may perform. These structures are represented by graphs, containing goals and subgoals that can be achieved, along with the methods required to achieve them. Each agent has its own graph, and tasks can be shared between graphs, creating relationships where negotiation or coordination may be of use. The TAEMS framework does no explicit planning, but provides some scheduling techniques. Its focus is on coordinating tasks of agents where specific deadlines may be required. The development has been discontinued since 2006.

Machinetta [112] is a coordination framework that uses a proxy-based integration infrastructure for coordinating teams of heterogeneous entities such as robots, agents, and persons. The use of proxies allows reusable teamwork capabilities across different domains and high-level team-oriented programming as lightweight Java processes. Proxies serve to initiate and terminate team plans, failure handling, sharing information, and coordination tasks allocation. It has no explicit planning component, but uses scheduling techniques to coordinate tasks. Machinetta is no longer in development.

Magentix2[4] [121], an upgraded version of the original Magentix [1], provides support in three different abstraction levels: *i)* organisation level, with technologies and techniques related to agent societies; *ii)* interaction level, with technologies and techniques related to communications between agents ; and *iii)* agent level, with technologies and techniques related to individual agents. These are provided by multiple building blocks that can be used for the development of MAS such as communication infrastructure, tracing, conversational agents, argumentative agents, Jason agents, and a HTTP interface.

JaCaMo[5] [13] is composed of three technologies, Jason, CArtAgO, and Moise, each representing a different abstraction level. Jason is used for programming the agent level, CArtAgO is responsible for the environment level, and Moise for the organisation level. JaCaMo integrates these three technologies by defining a semantic link among concepts in different levels of abstraction (agent, environment, and organisation). The end result is the JaCaMo MAS development platform. It provides high-level first-class

[3] http://www.activecomponents.org/
[4] http://www.gti-ia.upv.es/sma/tools/magentix2/
[5] http://jacamo.sourceforge.net/

support for developing agents, environments, and organisations, allowing the development of more complex multi-agent systems.

The Multi-Agent Programming Contest (MAPC) is an annual international competition, initiated by Jürgen Dix, Mehdi Dastani, and Peter Novak in 2005. Its purpose is to stimulate research in multi-agent programming by introducing complex benchmark scenarios that required coordinated action and can be used to test and compare multi-agent programming languages, platforms, and tools. The winners of MAPCs in 2013 [151], 2014[6], and in 2016 [28] used JaCaMo to program their agents. Thus, JaCaMo's performance and its multiple abstraction levels were the reasons why we chose it to implement our distributed online multi-agent planning framework. A more detailed discussion about JaCaMo and its functionalities is presented in the next section.

2.2.3 JaCaMo

JaCaMo[7] [13] is a MAS development platform that explores the use of three MAS programming dimensions: agent, environment, and organisation. According to the authors of [13], "the agent-based development of complex distributed systems requires design and programming tools that provide first-class abstractions along the main levels or dimensions that characterise multi-agent systems". A JaCaMo MAS (i.e., a software system programmed in JaCaMo) is given by an organisation specification in Moise, wherein autonomous agents programmed in Jason can assume roles in the organisation, and work in a shared distributed artifact-based environment programmed in CArtAgO.

Jason [15] handles the agent dimension. It is an extension of the AgentSpeak(L) language [100], based on the BDI agent architecture. Agents in Jason react to events in the system by executing actions on the environment, according to the plans available in each agent's plan library. One of the extensions in Jason is the addition of Prolog-like rules to the belief base of agents.

CArtAgO [104] is based on the A&A (Agents and Artifacts) model [96], and manages the environment dimension. Artifacts are used to represent the environment, storing information about the environment as observable properties (beliefs that are added to the agent's belief base) and providing actions that can be executed through operations. Agents can focus on specific artifacts in order to obtain information contained on that artifact, agents then receive the observable properties as beliefs and are able to execute the artifact's operations.

Moise [65] operates the organisation dimension, regulating the specification of organisations in the MAS. Moise adds first-class elements to the MAS such as roles,

[6] https://multiagentcontest.org/2014/
[7] http://jacamo.sourceforge.net/

groups, social goals, missions, and norms. Agents can adopt roles in the organisation, forming groups and subgroups. Missions are defined to achieve social goals. The behaviour of agents that adopt roles to accomplish these missions is guided by norms.

An overview of how JaCaMo combines these different levels of abstraction and how they interact with each other can be observed in Figure 2.6 [13]. In the top-most level, the organisation dimension is composed of a scheme, a set of missions, and a set of roles. These roles are adopted by agents that inhabit the agent dimension. In the bottom-most dimension, the environment houses artifacts that relate information about the environment, grouped into workspaces that agents can access. These workspaces can be distributed across multiple network nodes, effectively distributing the MAS.

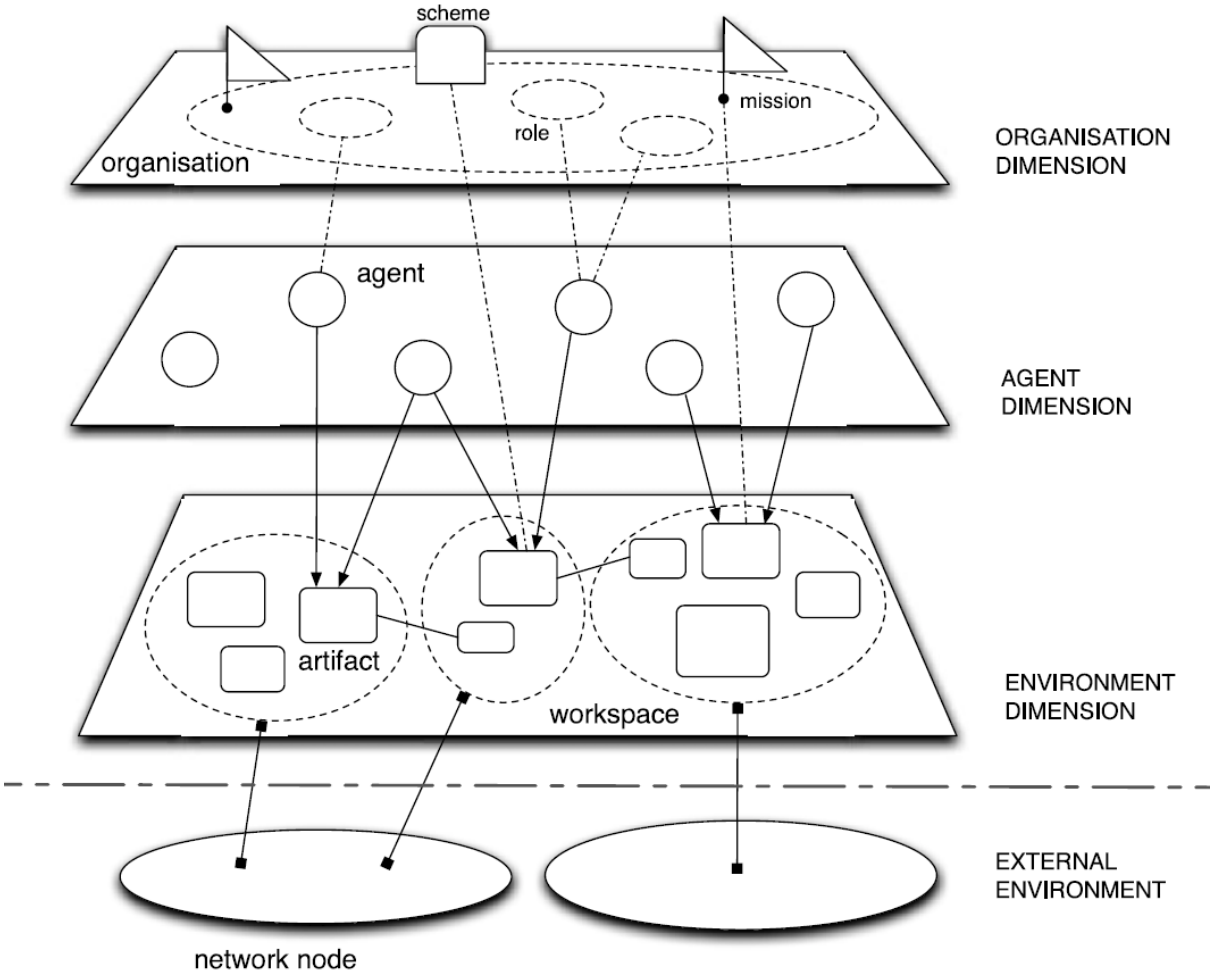

Figure 2.6 – The JaCaMo MAS development platform overview [13].

JaCaMo integrates these three technologies by defining a semantic link among concepts of the different abstractions (agent, environment, and organisation) at the meta-model and programming levels, in order to obtain a uniform and consistent programming model that simplifies their combination for the development of MAS. This platform provides high-level first-class support for developing agents, environments, and organisations, allowing for the development of more complex MAS.

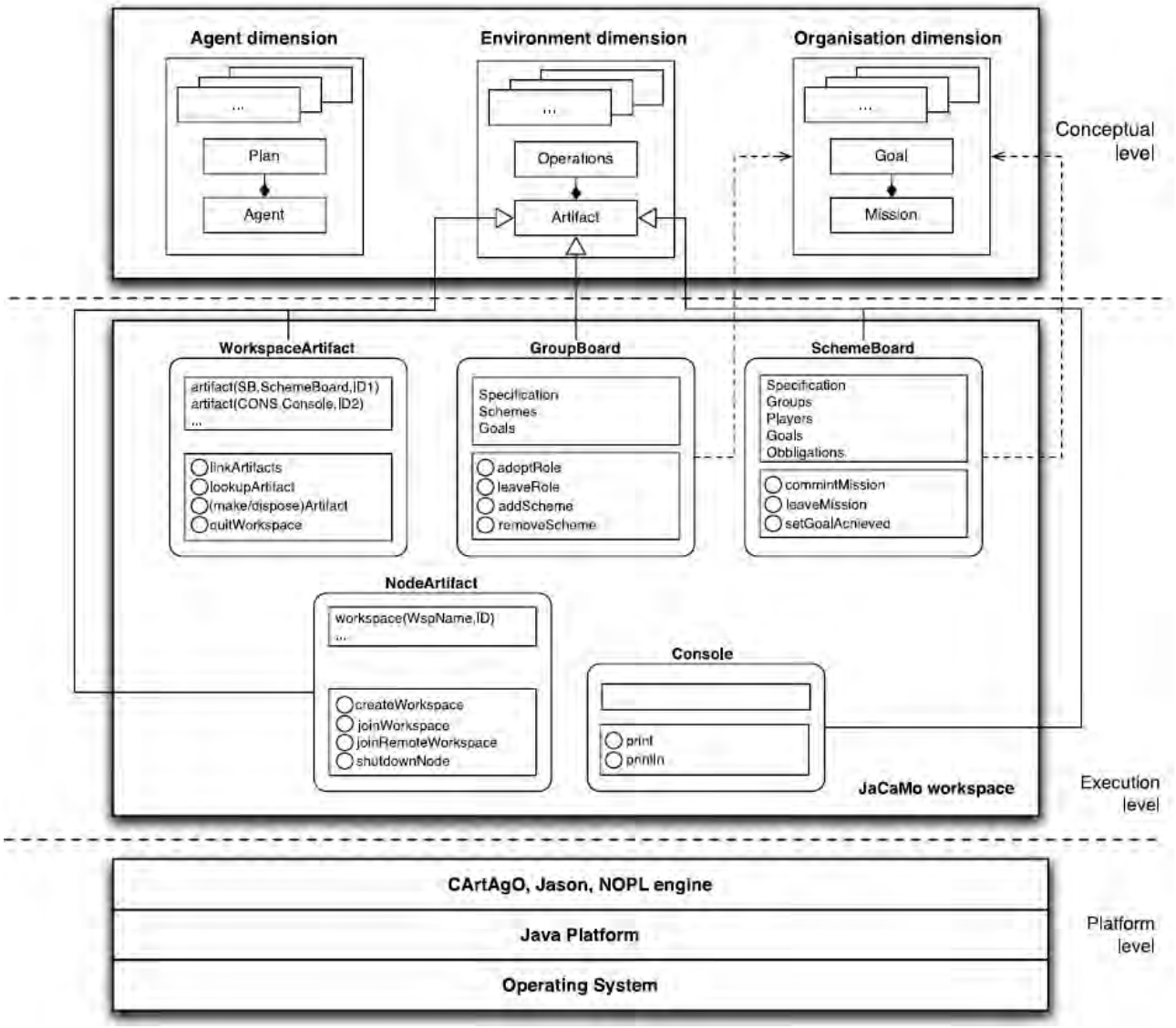

Figure 2.7 – JaCaMo runtime model and the standard set of artifacts available [13].

The structure of a JaCaMo MAS runtime is shown in Figure 2.7 [13]. Starting at the platform level, above the operating system and the Java platform, are CArtAgO, Jason, and NOPL engine (Moise implementation). At the execution level, we have several standard artifacts that are executed in every JaCaMo MAS application. The upper rectangle in artifacts contain the list of observable properties, while the bottom rectangle shows the available operations. For example, the *GroupBoard* has the *Specification*, *Scheme*, and *Goals* of an organisation's group listed as observable properties. Agents that focus on this artifact will gain access to this information, and will be able to execute the actions (operations) *adoptRole*, *leaveRole*, *addScheme*, and *removeScheme*. At the conceptual level we have the dimensions previously discussed in Figure 2.6.

Before we go into detail on each of JaCaMo's technologies, consider an example that we use to show code snippets for each technology.

JaCaMo MAS example — *Build-a-House*

We use the *Build-a-House* example that comes with the JaCaMo distribution, also described in [13]. In this JaCaMo MAS example, an agent called *Giacomo* wants to build a house on a piece of plot that he owns. First, he will have to hire various specialised companies, each represented by a company agent. Second, *Giacomo* needs to ensure that the various tasks required to build the house are coordinated and executed by the contractors.

In the first phase, *Giacomo* has to hire one company for each of the several tasks involved in building the house, such as site preparation, laying floors, building walls, and building the roof. An auction-based mechanism is used by *Giacomo* to select the best company for each of the tasks — the best company is the one that offers the cheapest service for a specific task. *Giacomo* starts an auction for a task with the maximum price that he is willing to pay. Companies that are suitable for the job can place bids. After a certain deadline, *Giacomo* selects the lowest bid at the time that the auction ended, and awards the contract for that task to the company that placed the bid.

In the second phase, after companies have been hired, the contractors have to execute their assigned tasks. Some kind of coordination needs to be employed here, since tasks can depend on other tasks being completed first, and some tasks can be done in parallel. This coordination is done via social schemes in the Moise organisation.

The *.jcm* file contains the initial configuration used to start the MAS, defining the names and number of agents that will be created, the artifacts, and the organisation specification. In this file it is also possible to define groups, assign initial roles to agents, and much more. In Listing 2.2, we show the *.jcm* for the *Build-a-House* example, limiting it to agent definition only in order to keep it simple. *Giacomo* is the agent that

Listing 2.2 – JaCaMo .jcm code snippet for the *Build-a-House* example.

```
1 mas house_building {
2   agent giacomo
3   agent companyA
4   agent companyB
5   agent companyC {
6     instances: 5
7   }
8 }
```

wants to build a house, and the others are company agents, with *companyC* creating five `instances` of the agent (*companyC1*,...,*companyC5*).

Agent Abstraction Level — Jason

Jason[8] [15] is an agent-oriented programming language for the development of MAS based on the BDI architecture. Jason is an implementation of the AgentSpeak language, with some new and unique extensions. AgentSpeak, initially conceived by Rao [100], represents an abstraction of implemented BDI systems like PRS which allows agent programs to be written and interpreted similarly to horn-clause logic programs, hence the resemblance to the logic programming language Prolog. A plan in AgentSpeak is triggered when events (internal or external) occur, followed by checking if the context of the plan is applicable; the body of the plan is composed of basic actions and/or subgoals that need to be achieved in order for the plan to be successful.

In Jason, an agent is an entity composed of a set of *beliefs*, representing the agent's current state and knowledge about the environment in which it is situated; a set of *goals*, tasks that the agent aims to achieve; a set of *intentions*, tasks that the agent is committed to achieve; and a set of *plans*, courses of *actions* that are triggered by *events*. Events in Jason can be related to changes in the agent's belief base, or to the addition or removal of goals. The agent reacts to these events by creating new intentions, provided that there are applicable plans for reacting to that event.

Thus, an agent program is a set of initial goals, beliefs, rules, and plans. Goals are preceded by `!`, plans assume the form `te : c <- b`, where `te` is a triggering event that the plan can react to; `c` is the context (optional), a conditional formula that must be true for the plan to be applicable; and `b` is the body of the plan, a sequence of actions and subgoals.

In Figure 2.8 [15], the reasoning cycle of a Jason agent is illustrated. When an agent receives a perception from the environment, it goes through the BRF (Belief Revision Function in step 1) that adds, update, or remove, a belief from the belief base.

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

This causes an external event to be generated and added to the set of events (in step 2). An event is chosen and unified with all relevant plans from the plan library (step 3). The context of all relevant plans are tested (step 4), and the ones which passed are separated as applicable plans (step 5) and added to the stack of intentions. An intention from the stack of intentions is chosen (step 6), and then its corresponding action is executed (step 7).

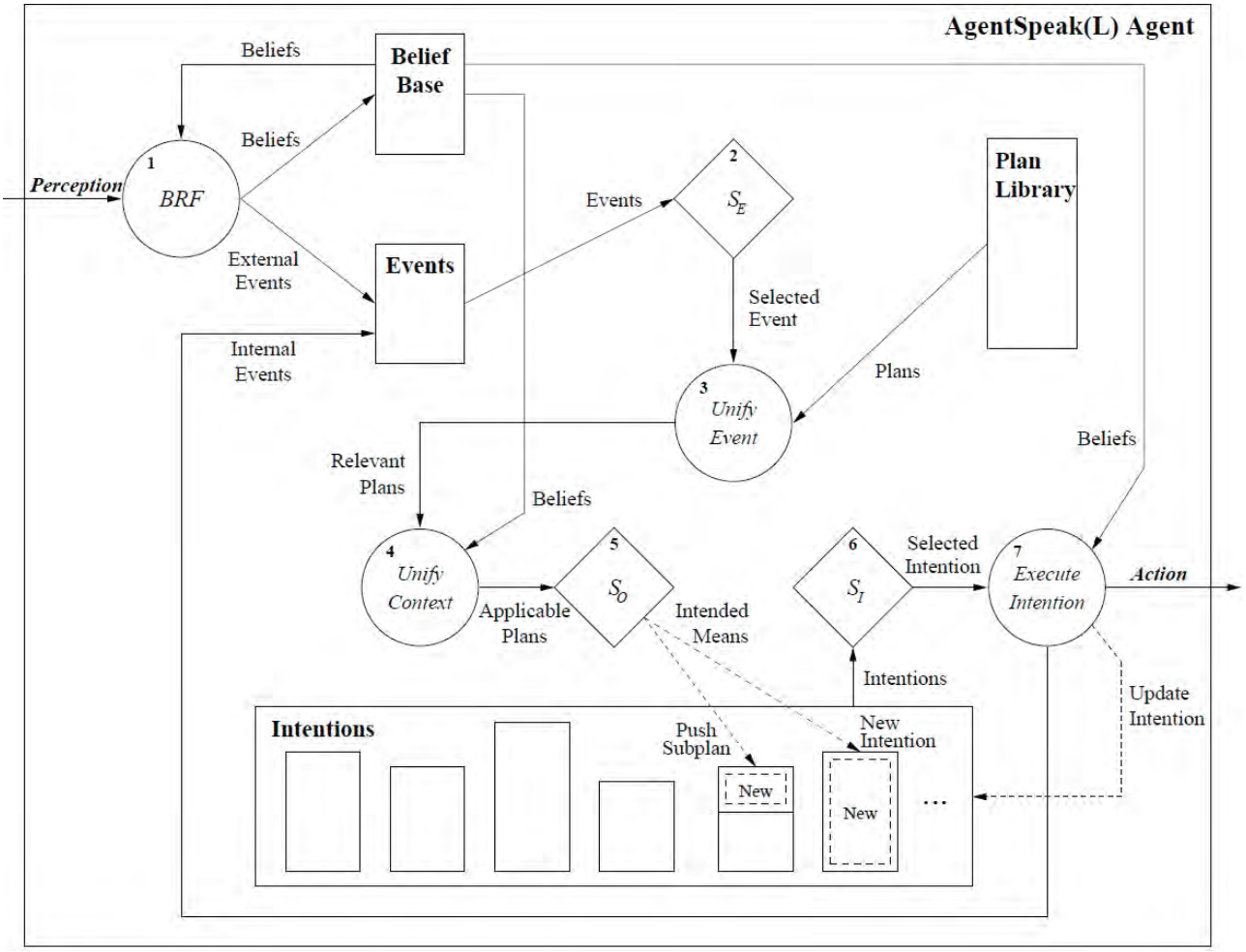

Figure 2.8 – Jason overview [15].

Listing 2.3 shows a snippet of the *Giacomo* agent code from the *Build-a-House* example. The initial goal `!have_a_house` creates a goal addition event as soon as *Giacomo* is created. The plan `+!have_a_house` has an empty context, and thus, immediately reacts to this event by adding two additional subgoals, `!contract` to initiate the contract phase, and `!execute` to build the house. By default, these subgoals are triggered in the order that they appear, that is, the contract subgoal has to be completed first, and then goal execute is added. The `+!contract` plan adds a subgoal to create auction artifacts for the tasks to be auctioned, and then adds a subgoal to wait for bids from company agents. The `+!wait_for_bids` plan (line 15) uses the `.wait` internal action to wait 5000 milliseconds, and then adds a subgoal to show the winners of

Listing 2.3 – Agent code snippet for the *Build-a-House* example.

```
1  !have_a_house.
2
3  +!have_a_house
4  <-
5      !contract;
6      !execute.
7
8  +!contract
9  <-
10     !create_auction_artifacts;
11     !wait_for_bids.
12
13 +!wait_for_bids
14 <-
15     .wait(5000);
16     !show_winners.
17
18 +!execute
19 <-
20     makeArtifact("hsh_group","GroupBoard",["house-os.xml", house_group],GrArtId);
21     adoptRole(house_owner)[artifact_id(GrArtId)];
22     focus(GrArtId);
23     !contract_winners("hsh_group");
24     makeArtifact("bhsch", "SchemeBoard",["house-os.xml", build_house_sch], SchArtId);
25     focus(SchArtId);
26     ?formationStatus(ok)[artifact_id(GrArtId)];
27     addScheme("bhsch")[artifact_id(GrArtId)].
```

each auction. The `+!execute` plan creates the organisational artifacts *GroupBoard* and *SchemeBoard*, focuses on both, adopts the role of *house_owner*, waits until all companies have adopted their roles, and finally, adds the scheme that will coordinate the execution to build the house.

Environment Abstraction Level — CArtAgO

CArtAgO[9] [104] is a framework that provides the infrastructure for environment-oriented programming and execution in multi-agent systems. The underlying idea is that the environment can be used as a first-class abstraction for designing MAS. It serves as a computational layer that encapsulates functionalities and services that the agents can explore at runtime. CArtAgO is based on the A&A meta-model [96]. In this model, environments are designed and programmed as a dynamic set of computational entities called artifacts, dispersed into several workspaces, and possibly distributed among various nodes of a network.

Artifacts in CArtAgO are programmed using the Java language. The current state of the environment is represented by observable properties. When an agent focuses

[9] http://cartago.sourceforge.net/

on a particular artifact, it gains access to the observable properties of the artifact, that are directly represented as beliefs in that agent's belief base. If the value of an observable property is changed (e.g., another agent executed an action that caused a change in the environment), the belief base of all agents that have focused on that artifact are updated accordingly. This process is shown in Figure 2.9 [103].

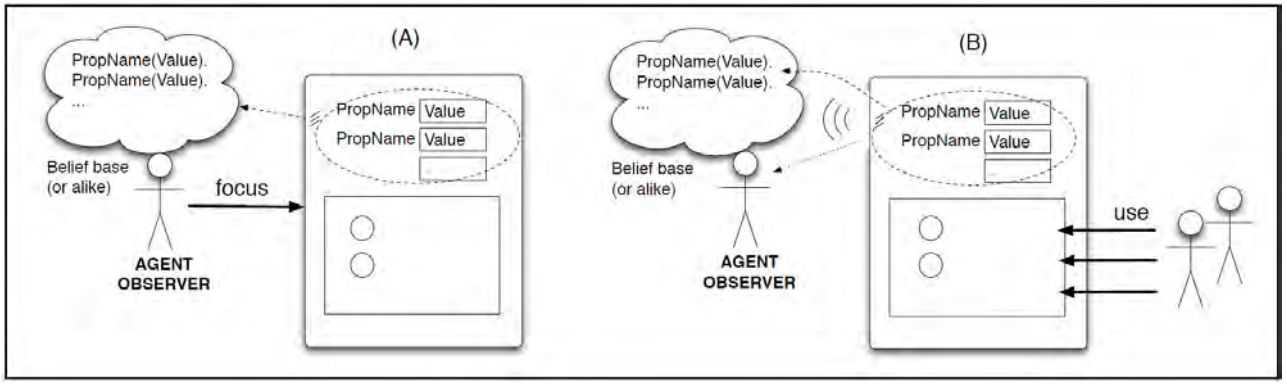

Figure 2.9 – Overview of how focus works [103].

The agent's actions in the environment are mapped as operations in the artifacts. Operations can cause changes in the observable properties that dynamically update the belief base of all agents that are observing the artifact. As long as an agent focuses on an artifact, he will have access to that artifact's operations and will be able to carry out the actions associated with it. The use of operations is depicted in Figure 2.10 [103].

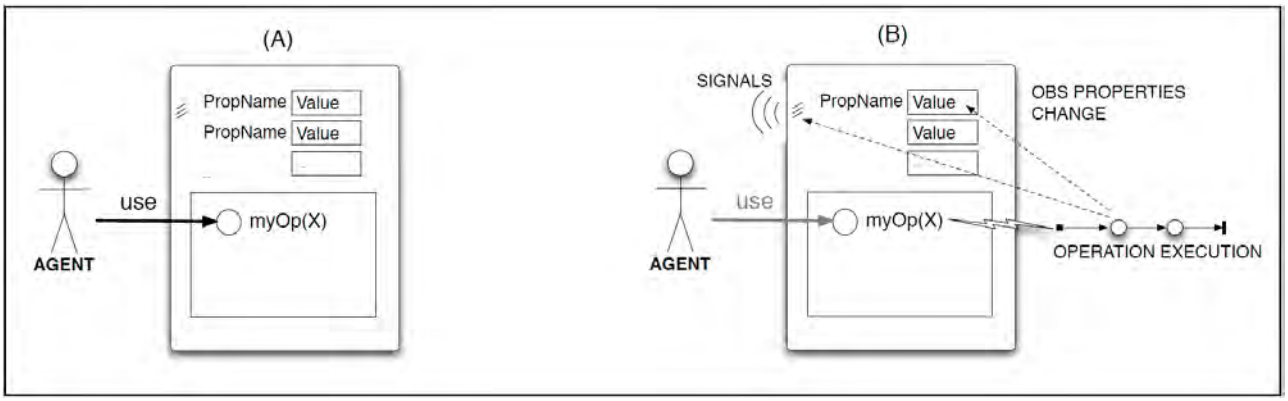

Figure 2.10 – Using operations in artifacts [103].

In Figure 2.11 [103], the A&A meta-model of CArtAgO is shown. A workspace can represent a place in the world, a location where one or multiple activities involving a set of agents and artifacts can happen. The execution of an operation can also generate signals that are perceived by agents. Differently from observable properties, signals are useful to represent non-persistent observable events that occurred, conveying some

useful information to the agents. Artifacts can be linked together, in which case an artifact can execute the operations of another linked artifact. Linked operations cannot be accessed by agents, but only by artifacts that are linked together. Finally, an artifact can be equipped with a manual, a document that can be consulted by agents, with a description of the resources and tools provided by the artifact.

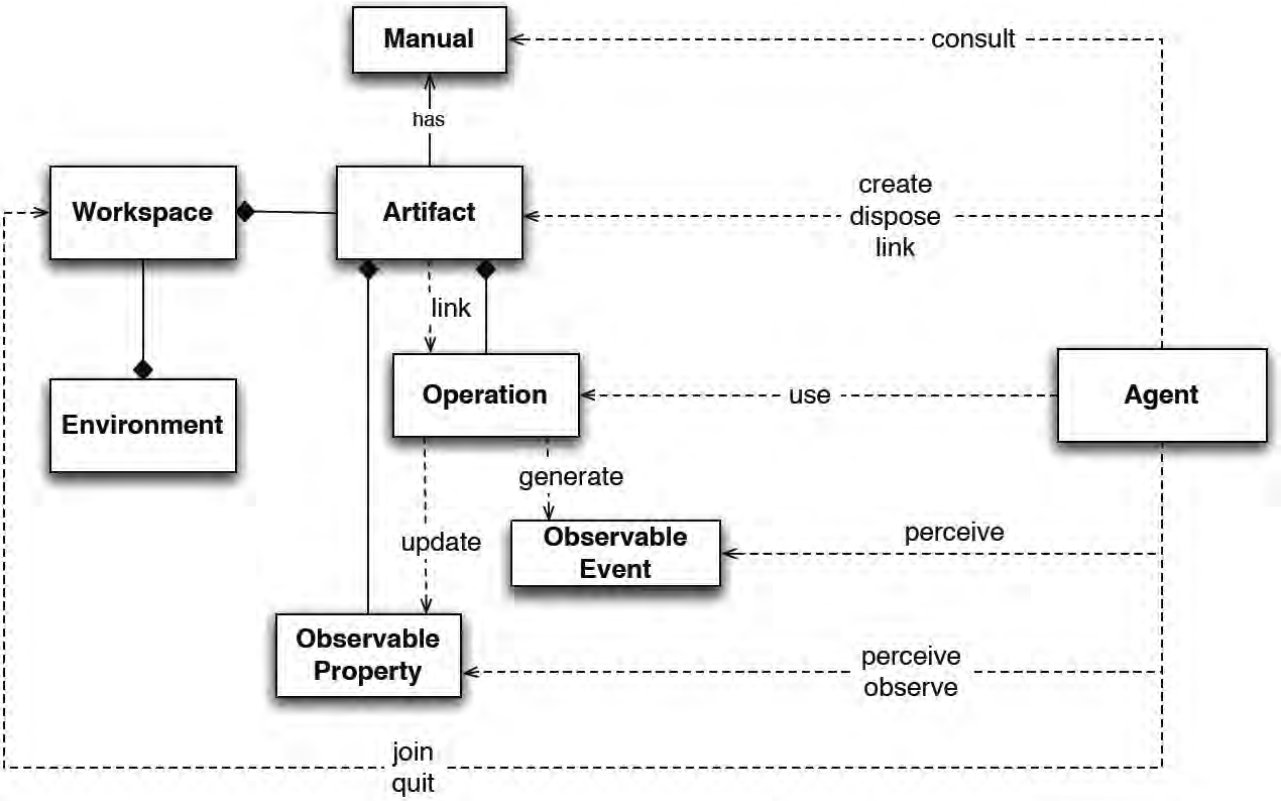

Figure 2.11 – CArtAgO A&A meta-model [103].

Listing 2.4 shows a code snippet for the *AuctionArt* artifact from the *Build-a-House* environment. This artifact represents the auction-based mechanism that is used by *Giacomo* to select the best company for each of the tasks, and it is also used by company agents to place bids to a task. The *init* function is where initial values for observable properties can be assigned.

Four observable properties are created for this artifact: *task*, the task description; *maxValue*, maximum bid value that agent *Giacomo* is willing to pay; *currentBid*, value of the currently lowest bid, it starts with the maximum possible value; and *currentWinner*, stores the name of the agent with the best bet. The *AuctionArt* artifact also has the *bid* operation. Company agents that focus on this artifact are able to execute the *bid* action for that specific task. The *bid* operation retrieves values from *currentBid* and *currentWinner* observable properties and update their values if the *bidValue* proposed by the agent is lower than the current best value (lines 12–15).

Listing 2.4 – Environment auction artifact for the *Build-a-House* example.

```
1  public class AuctionArt extends Artifact {
2    public void init(String taskDs, int maxValue) {
3      defineObsProperty("task",        taskDs);
4      defineObsProperty("maxValue",    maxValue);
5      defineObsProperty("currentBid",  maxValue);
6      defineObsProperty("currentWinner", "no_winner");
7    }
8    @OPERATION
9      public void bid(double bidValue) {
10     ObsProperty opCurrentValue  = getObsProperty("currentBid");
11     ObsProperty opCurrentWinner = getObsProperty("currentWinner");
12     if (bidValue < opCurrentValue.intValue()) {
13       opCurrentValue.updateValue(bidValue);
14       opCurrentWinner.updateValue(getOpUserName());
15     }
16   }
17 }
```

Organisation Abstraction Level — Moise

Moise[10] [65] includes a specification language, a management infrastructure, and support for organisation based reasoning mechanisms at the agent level. The organisation is specified in an eXtensible Markup Language (XML) file, divided in three sections: *structural specification*, where groups, roles, and links between roles are specified; *functional specification*, where schemes are specified containing a group of goals and missions, along with ordering constraints for goals establishing which have to be executed in parallel and which have to be executed in sequence; and *normative specification*, where obligations and permissions of missions can be set to certain roles.

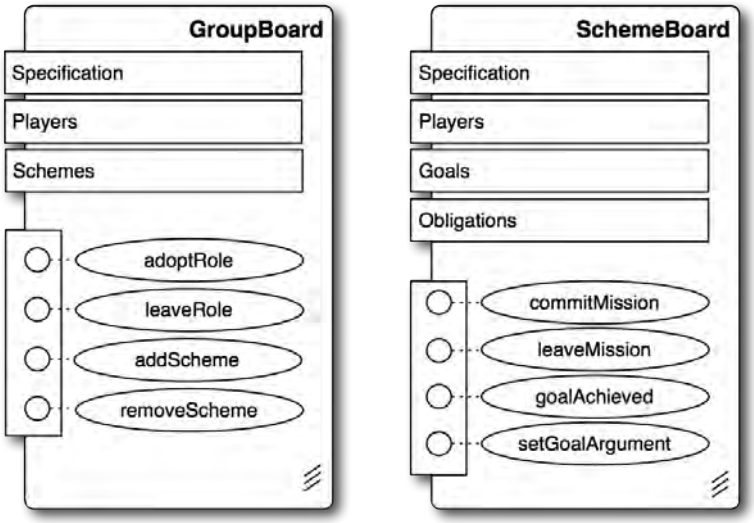

Figure 2.12 – Moise GroupBoard and SchemeBoard artifacts [13].

[10] http://moise.sourceforge.net/

The management infrastructure of organisations in JaCaMo works through CArtAgO artifacts, another example of the synergistic behaviour between JaCaMo components. These organisational artifacts are used to help coordinate agents in order to achieve their social goals, and also control the adoption of roles, group creation, among others. These artifacts are illustrated in Figure 2.12 [13]. The *GroupBoard* was already previously covered. The *SchemeBoard* is responsible for managing social schemes, or goal trees. In these schemes, goals are attributed to missions, which are then assigned to roles.

In Listing 2.5 we show a code snippet with a few lines from each Moise specification for the *Build-a-House* MAS example. Moise contains many more useful concepts (e.g., links, formation constraints, etc.) that can be used, which we omit here for simplicity, check the full code of this example in the JaCaMo distribution. In the structural specification, the top role *house_owner* is defined, while roles *plumber* and *painter* are defined as extensions of the *building_company* top role. In the functional specification, a scheme for building the house determines that goals *plumbing_installed* and *exterior_painted* can be executed in parallel, and only when both are finished can the achievement of goal *interior_painted* proceed (as determined by the sequence operator in line 10). Also in the functional specification, goals are attributed to missions which can have cardinalities. Finally, in the normative specification, a norm of type *obligation* is set connecting the mission *paint_house* to the role of *painter*.

Listing 2.5 – Organisation code snippet for the *Build-a-House* example.

```
1  <structural-specification>
2  <role-definitions>
3   <role id="house_owner" />
4   <role id="building_company" />
5    <role id="plumber" >       <extends role="building_company"/> </role>
6    <role id="painter" >       <extends role="building_company"/> </role>
7
8  <functional-specification>
9   <scheme id="build_house_sch">
10     <plan operator="sequence">
11       <plan operator="parallel">
12        <goal id="plumbing_installed"      />
13        <goal id="exterior_painted"       />
14       </plan>
15      <goal id="interior_painted"             />
16
17    <mission id="paint_house" min="1" max="1">
18     <goal id="exterior_painted" />
19     <goal id="interior_painted" />
20    </mission>
21
22 <normative-specification>
23 <norm id="n10"  type="obligation"  role="painter"       mission="paint_house" />
```

2.3 Related Work

This section describes some important and relevant research in automated planning, including recent multi-agent planners and related research in single-agent planning. As described in Section 2.1, automated planning can have many classifications. As such, there have been several surveys over the years describing advancements in particular areas of planning. Of interest and related to this research there are, for example: in [46], a survey on distributed online planning is presented, with the state of the art in distributed and online planning at the time (1999), and a design for a distributed online planning paradigm; a survey [80] that presents a collection of recent techniques (2013) used to integrate single-agent planning in BDI-based agent-oriented programming languages, focusing mostly on efforts to generate new plans at runtime; and a multi-agent planning survey in 2009 [43], describing common multi-agent planning stages and listing approaches from the literature used in each stage, or even in a combination of multiple stages.

To make the description of related work unambiguous, we classify them into three different sections: online single-agent planning, offline multi-agent planning, and online multi-agent planning. Although the work presented in this thesis is based on online multi-agent planning, many of its components can be seen in both online single-agent planning and offline multi-agent planning.

2.3.1 Online Planning

Besides CANPLAN, already discussed in previous sections, there are several other approaches that try to integrate planning into agent-programming languages. A modification of the X-BDI agent model [84] is presented in [82] that describes the relationship between propositional planning and means-end reasoning of BDI agents. Graphplan was used as the planning algorithm, and applied to a scenario of a production cell controlled by agents.

RETSINA [123] is a multi-agent system infrastructure that contains four different types of agents: *interface agents* that interact with users, *task agents* to formulate and carry out plans, *information agents* that store and provide information from various sources, and *middle agents* to match agents that require services with those that provide. Each agent has four modules for communicating, planning, scheduling, and monitoring execution of tasks. Differently from DOMAP, RETSINA does not provide any domain-independent goal allocation mechanism, they plan for pre-allocated goals. Its development has been discontinued.

In [81], by extending the AgentSpeak(L) interpreter, agents are able to call a classical planner to create new plans at runtime, in order to respond to unforeseen circumstances at design time when achieving its goals. Annotating the expected effects of plans at design time, allows them to be used during the creation of new plans. The translation between the planner's world view and the agents' action theory is very limited, effectively losing information that could potentially be used in planning and execution.

Markov Decision Process (MDP) [11] are used for modelling non-deterministic environments, providing value-oriented methods to coordinate agents. These methods deal with settings where action's outcomes are uncertain, and can even be extended to deal with partially observable environments, known as Partially Observable Markov Decision Processes (POMDP) [119]. For example, in [146], an online algorithm for planning under uncertainty in multi-agent settings that are modelled as decentralised POMDPs, requiring little to no communication. While in [18], qualitative decentralised POMDP is proposed as a a qualitative, propositional model for multi-agent planning under uncertainty with partial observability. This model is based on classical planners, allowing it to handle multi-agent planning problems significantly larger than other decentralised POMDP algorithms. And in [126, 125], Earley graphs are used to shown that it is possible to bridge the gap between HTNs and MDPs, allowing the use of probabilistic HTN for planning in an MDP environment. Although we do not cover non-deterministic planning, the existence of a bridge between HTNs and MDPs could facilitate an extension of DOMAP to allow multi-agent planning in these types of domains.

IndiGolog [57], an extension of Golog [76], is a programming language for autonomous agents that opted not to use classical planning, based on the premise that it often ends up being computationally costly. Instead, they used a high-level program execution [75] as a middle ground between classical planning and agent programming. The general idea of this approach is to give the programmer more control over the search effort: if a program is almost deterministic then very little search effort is required; on the other hand, the more a program has non-determinism, the more it resembles traditional planning. Because this language is not BDI-based, plans are not associated with goals and events, which, for example, makes it difficult to find an alternative plan when a selected plan fails.

The planning extension of the GOAL [60, 61] agent oriented programming language allows it to be used as a planning formalism as well. They show that a PDDL subset containing axioms, the action description language, and plan constraints can be compiled into GOAL and serve as an interface between BDI agent programs and planners. However, it is not clear how it compares against traditional PDDL planners, since no quantitative or qualitative comparison was shown.

These approaches were directed to single-agent planning in the context of agent-oriented programming languages, but they do not address the features found in multi-agent planning.

2.3.2 Offline Multi-Agent Planning

Kovacs proposed an extension for PDDL3.1 that enables the description of multi-agent planning domains and problems [71]. This extension of the formalism covers planning for agents in temporal, numeric domains. It is specifically designed to cope with the exponential increase in the number of actions, and the constructive and destructive synergies found in concurrent actions. Only the formalism is provided, it has not yet been implemented in any planner/system.

MAP-POP (Multi-Agent Planning based on Partial Order Planning) [74, 129, 130] is a cooperative multi-agent planner. It uses plan-space search, where agents consecutively refine an initially empty partial plan, until arriving at a solution. POP-based planners [4] work over all the planning goals simultaneously, maintaining partial order relations between actions without compromising a precise order among them until the plan's own structure determines that it is necessary. The model builds upon the concept of refinement planning [69], where agents propose successive refinements to a base plan until a consistent joint plan that solves the problem is obtained.

The MAP-POP algorithm, as described in [74, 129, 130], starts with an initial communication stage in which agents exchange information in order to generate data structures that will be used during planning. The planning itself is composed of two different stages that are interleaved: an internal planning process, where agents refine the current base plan individually with an internal POP system; and a MAS coordination process, where agents exchange the refinement plans generated and collectively select the next base plan to be refined. These stages repeat themselves until a solution is found. As the complexity of the problem increases, so does the time required for finding a solution, since the number of plans required to be refined, and the number of messages exchanged, increases. This results in a noticeable increase in planning time, especially when increasing the number of agents, as communication takes a considerable amount of time.

FMAP [131], an iteration of MAP-POP, is a general-purpose multi-agent planner with an advanced notion of privacy that was lacking from MAP-POP. Like MAP-POP, it uses an internal POP procedure to calculate all possible ways to refine a plan, and agents exchange plans and their evaluations by communication. In FMAP, this communication is governed by a coordinator agent. Agents will only communicate relevant information that they believe they can share with the rest of the agents, in order main-

tain privacy. Similarly to MAP-POP, the communication overhead from partial order planning in multi-agent settings does not scale very well when dealing with a large number of agents.

Another POP-based planner, FLAP [108] is a hybrid planner that combines partial-order plans with forward search and uses state-based heuristics. FLAP implements a parallel search technique that diversifies the search. Unlike the other planners, FLAP exploits delaying commitment to the order in which actions are applicable. This is done to achieve flexibility, reducing the need of backtracking and minimizing the length of plans by promoting parallel execution of actions. These changes come at an increase in computational cost, though it allows FLAP to solve more problems than other partial-order planners.

In [17], the CSP+planning methodology is introduced, separating public and private aspects of planning problems. This methodology is used to quantify the coupling level of a multi-agent planning problem, and facilitate coordination during planning. It quantifies coupling by using two parameters: the tree-width of the agent interaction graph; and the number of coordination points required per agent. Planning-First [94] is a general, distributed MAP algorithm, based on the CSP+planning methodology and the MA-STRIPS [17] planning formalism. Planning-First uses Distributed Constraint Satisfaction Problem (DisCSP) [150] to coordinate agents, and the Fast-Forward planning system [63] for local planning.

Using DisCSP in Planning-First was too much of a performance hindrance, thus the authors developed a new iteration called MAD-A* [92]. It is a multi-agent planning adaptation of one of the most well-known search algorithms, A*. By distributing the work among different agents, it removes symmetries and reduces the overall workload. Moreover, it reduces exactly to A* when there is only a single agent. MAD-A* uses heuristics to help coordinate agents during planning, and a multi-agent extension of the Fast Downward [59] planner. These different options for coordination mechanism and individual planner achieved a much better performance than Planning-First had obtained.

In [93], the authors propose a heuristic forward search for classical multi-agent planning that respects the natural distributed structure of the system, preserving agent privacy. According to the experiments, their system had the best performance (in most cases) in regards to planning time and communication, as well as the quality of the solution, when compared to other offline multi-agent planning systems. They can deal rather well with tightly-coupled domains, but add a communication overhead that slows their planning time performance on loosely-coupled domains.

CODMAP-15 Planners

In this section we describe four state-of-the-art multi-agent offline planners that participated in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP-15) [138]. These are the planners that we use to compare our approach in the experiments reported in Chapter 6. An extension of MA-PDDL [71], based on privacy notions from MA-STRIPS [17], was used as the official planning formalism[11] for the competition. This extension focuses on two concepts: *factorisation* separates domain information available for each agent, effectively limiting which STRIPS actions agents can use; and *privacy* determines which STRIPS facts are private, i.e., it is not affected and cannot affect more than one agent. The input accepted by planners had two versions: factored description, allows the definition of a separate domain and problem for each agent; and unfactored description, allows the definition of factorised privacy in a single domain and problem description.

SIW+ -then-BFS(f) [85] was the top performing planner with regards to planning time (and second overall when considering all metrics), out of a total of 12 planners (and 17 configurations). It is a single-agent agile planner that converts the multi-agent domain into a classical planning domain in such a way that privacy is respected. Thus, planning is done using a centralised classical planner that encodes the factorisation and creates a classical planning problem containing all of the necessary information. Because the privacy of objects and fluents used in the competition's formalism are static, their encoding technique is based on only allowing valid grounding that respects privacy by modifying the domain description. It uses two planners, first it tries to solve the problem with SIW+, and if it fails to find a solution then BFS(f) is invoked. The goal behind this approach is to find a solution as fast as possible, while still being complete and sound.

CMAP-t [16] obtained second place (third overall), it starts with the compilation of MA-PDDL into normal PDDL, extracting agent types and private predicates. Then, it obfuscates the private parts of domains and problems, and assign goals to agents following the subset strategy. This strategy computes the relaxed plan for each goal and the subset of agents that appear as arguments of any action in each of them. These are the agents that could potentially be needed to solve the goal. After goals are assigned, CMAP-t merges all domain and problem representations from agents that were assigned goals into a single obfuscated file, and calls a single agent planner to solve it.

ADP-legacy [37, 35] achieved third place (first overall), and is a centralised single-threaded planning algorithm implemented in the Fast-Downward planning system [59]. ADP deduces agent decompositions from standard PDDL encodings and then generates a heuristic to exploit these decompositions for efficient planning, ignoring the

[11]The complete BNF definition of MA-PDDL with privacy can be found at http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF.pdf

agent factorisation and privacy from MA-PDDL. It decomposes the problem into an environment and a number of agents who can interact with it, but not directly effect each other. The heuristic used adapts the Fast-Forward [63] relaxation heuristic to consider multi-agent information. The legacy version adds some randomness in the search process through non-deterministic agent assignment in each run of the same problem.

PMR [77] obtained eighth place, but was different from other planners. PMR uses three planners: the first uses goal allocation followed by individual planning; if the resulting plan is empty, then it uses a centralised single-agent planner to attempt to solve it; and if the solution is still not sound, then another planner capable of applying plan reuse is used. PMR also translates from MA-PDDL to PDDL, and it obfuscates private parts, similar to CMAP-t.

2.3.3 Online Multi-Agent Planning

In [78], de-commitment penalties and a Vickrey auction mechanism are applied in an airport domain to solve the problem of de-icing and anti-icing aircrafts during winter. Agents in this domain are self-interested and often have conflicting interests. Performance on both of these mechanisms were positive, although they are restricted to this domain, since they use domain-specific knowledge.

The TAEMS framework [45], mentioned in Section 2.2.2, provides a high-level modelling language for describing the tasks that agents may perform. Although TAEMS does no explicit planning, we included it here since it allows for online scheduling, and tries to solve time-constraints often required in real-world applications.

In [31], multi-agent planning algorithms were proposed to exploit summary information to coordinate agents before planning. The authors claim that by associating summary information with abstract operators in plans, plan correctness can be ensured, while still gaining in efficiency. The general idea is to annotate each abstract operator with summary information about all of its potential needs and effects. This often resulted in an exponential reduction in planning time compared to a flat representation, but might lead to an exponential increase in the summary information size. Their approach depends on some specific conditions and assumptions, limiting it to certain domains.

A Multi-Agent Planning Language (MAPL) is proposed in [21]. Agents expressed in this language interleave planning, acting, sensing, and communication. Their approach is based on sharing knowledge in order to ensure the synchronous execution of joint plans. It is different from our approach, where agents keep their knowledge private and are coordinated by the organisation via social laws.

There have been several studies regarding the temporal decoupling problem in MAP. In this context, the temporal decoupling problem happens when agents that are collaborating on a set of temporally dependent tasks need to coordinate the execution of those tasks by applying additional temporal constraints in order to ensure that agents working on a different set of related tasks may operate independently. Techniques for temporal decoupling are especially useful for scheduling execution of tasks that involve management of resources at runtime.

In [133], an algorithm is used to determine the minimum number of resources an agent requires to accomplish its ground handling task in an airport application; later in [98], the Simple Temporal Problem (STP) is discussed, reporting that finding an optimal decoupling for STP is NP-hard, and that it can only be solved efficiently if all agents have linear valuation functions; in [143], the authors propose a characterisation of weakly-coupled problems and quantify the benefits of exploiting different forms of interaction structure between agents; and finally, in [68], a new algorithm is proposed for temporal decoupling and its efficiency is evaluated against other multi-agent STP algorithms; its results show that it still maintains the same computational complexity as the others, but also manages to surpass them in efficiency.

To the best of our knowledge there are no other recently implemented multi-agent online planner that takes advantage of the various programming dimensions in MAS, such as the ones available in JaCaMo.

In this chapter, we provided the necessary background and related work on automated planning and intelligent agents. In the next chapter, we introduce the Floods domain.

3. FLOODS DOMAIN

Past editions of International Planning Competitions (IPCs) have contributed with many robust single-agent planning domains and problems. Recently, the ICAPS (International Conference on Automated Planning and Scheduling) community has been exploring multi-agent planning further, as evidenced by the workshop series on Distributed and Multi-Agent Planning (DMAP). In the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP-15), held in conjunction with DMAP 2015, the organising committee described it as a possible test run for future IPCs focused on multi-agent planning.

The lack of robust and complex multi-agent domains led us to design a new domain, which we call *Floods* [24, 25, 22, 23], in order to best exploit the advantages of MAP and MAS. The inspiration for this specific domain came from the classical Rovers domain and a real-world scenario, on using computer science technology (e.g., a team of autonomous robots) to help mitigate and prevent natural disasters. This scenario is specifically targeted at search and rescue during flood disasters, often caused by intense hydro-meteorological hazards that can lead to severe economic losses, and in some extreme cases even deaths.

In the Floods domain, a team of autonomous robots are dispatched to monitor flood activity in a region with multiple *areas* that can be affected by floods. Social goals come from Centres for Disaster Management (CDM) that are located in the region being monitored. *Water samples* can be requested to be collected from certain areas, and during flood events *victims* may be detected and in need of *first aid kits*. There are three different types of autonomous vehicles operated by heterogeneous agents:

- **Unmanned Aerial Vehicle (UAV)**: are vehicles (e.g., drones) that are mostly used for taking aerial images of objectives. In our domain, UAVs are only used for capturing images, but are much faster than other vehicles for this task since they do not have any movement restrictions.

- **Unmanned Ground Vehicle (UGV)**: are vehicles that operate in contact with the ground and without an on-board human presence. In the Floods domain, UGVs are used for ground surveillance in areas that are connected via *ground paths*. They are also used to provide assistance to victims by transporting first aid kits to first responders located in areas affected by floods.

- **Unmanned Surface Vehicle (USV)**: are vehicles that operate on the surface of the water without a crew on-board. These vehicles are very useful to scout areas that other vehicles might not be able to go (or might be too risky/costly). In our domain, USVs can move through areas connected by *water paths*. Besides being

able to capture images, USVs also have the ability to collect water samples for the CDM, by using a variety of water sensors and actuators.

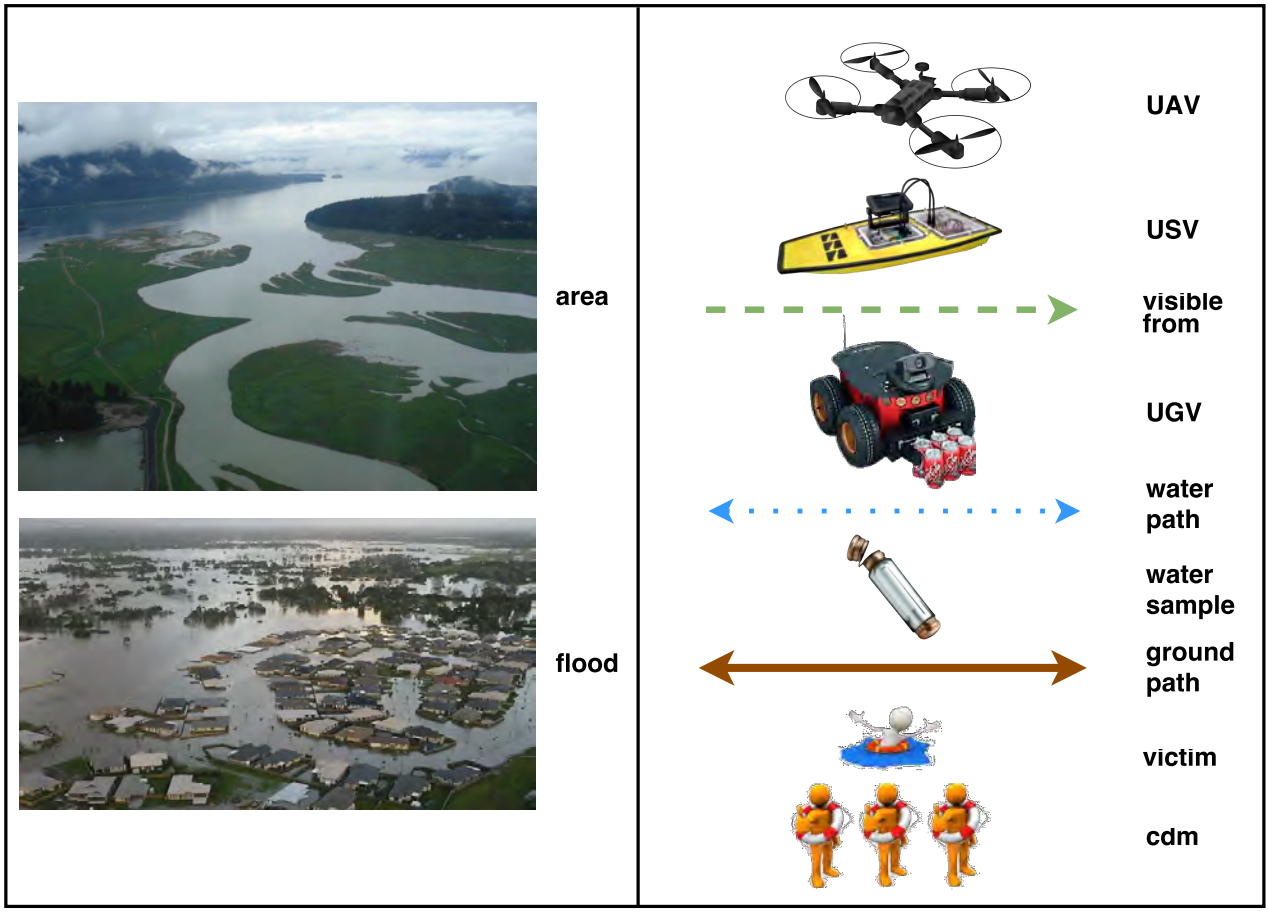

Figure 3.1 – Elements found in the Floods domain.

All of these elements from the Floods domain are illustrated in Figure 3.1. A simple problem in the Floods domain is represented in Figure 3.2. There are four areas in a region where flood events are common when under the effects of heavy rain. In this problem, there are four flood events occurring, *flood2* and *flood4* visible from *area4* and *flood1* and *flood3* visible from *area3*. There is a ground path between *area1* and *area3*. There are water paths between *area1* and *area2*, and between *area2* and *area4*. The *CDM* has established a base of operations in *area1*. The agents *uav1*, *ugv1*, and *ugv1* all start at *area1*, with the goals to take a picture of *flood1*, *flood2*, *flood3*, and *flood4*. This problem is used in the subsequent sections as a running example.

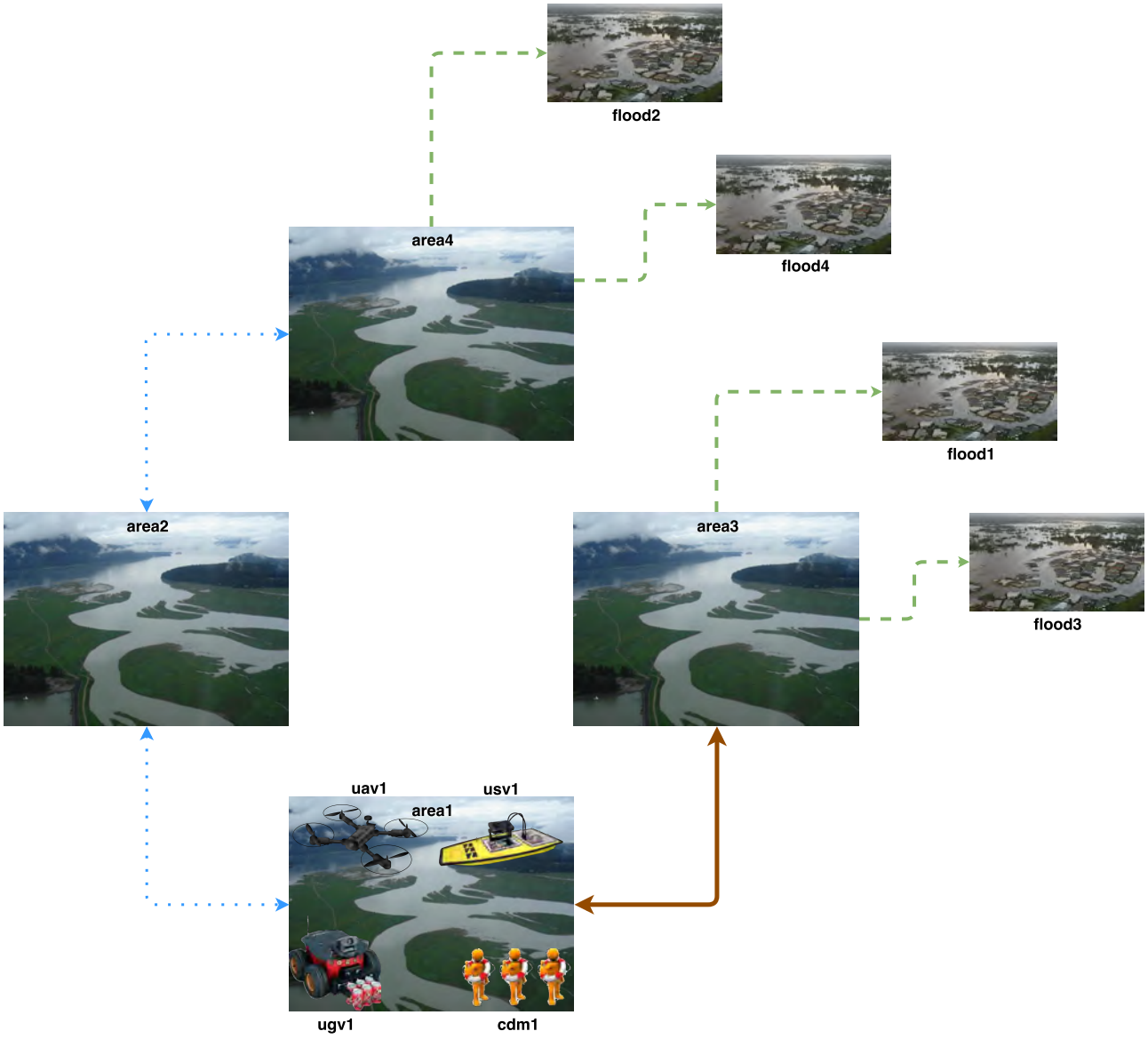

Figure 3.2 – A simple problem in the Floods domain.

3.1 HTN representation

In Listing 3.1, we show an example[1] of the floods UAV domain in SHOP2's HTN formalism. Because each agent has its own domain and problem representation, there is no need to specify which vehicle is going to be using an action. Thus, the layout is much cleaner than in usual planning formalisms, the planner has less conditional tests to do (slightly improving performance), and the search is effectively pruned since each planner will only expand their own agent's path, instead of trying all agents for all goals (greatly improving performance).

Listing 3.1 – HTN domain specification of an UAV agent.

```
1  (defdomain floods-uav (
2  (:operator (!navigate ?from ?to)
3        ( (area ?from) (area ?to) (at ?from) )
4        ( (at ?from) )
5        ( (at ?to) )
6  )
7  (:operator (!take_picture ?area ?disaster)
8        ( (area ?area) (disaster ?disaster) (visible_from ?disaster ?area) (at ?area) )
9        ( )
10       ( (have_picture ?disaster) )
11 )
12 (:operator (!communicate_data ?cdm ?disaster ?area1 ?area2)
13       ( (disaster ?disaster) (cdm ?cdm) (at ?area1) (cdm_at ?cdm ?area2) (area ?area1) (area ?area2) (
         have_picture ?disaster) (in_range ?area1 ?area2) )
14       ( (have_picture ?disaster) )
15       ( (communicated_data ?disaster) )
16 )
17 (:method (navigate ?to)
18       ( (at ?from) )
19       ( (navigate_uav ?from ?to) )
20 )
21 (:method (navigate_uav ?from ?to)
22       ( (at ?to) )
23       ( )
24       ( )
25       ( (!navigate ?from ?to) )
26 )
27 (:method (get_picture ?disaster)
28       ( (disaster ?disaster) (visible_from ?disaster ?area) )
29       ( (navigate ?area) (!take_picture ?area ?disaster) (send_data ?disaster) )
30 )
31 (:method (send_data ?disaster)
32       ( (disaster ?disaster) (have_picture ?disaster) (cdm_at ?cdm ?area2) (area ?area2) (in_range ?
         area1 ?area2) )
33       ( (navigate ?area1) (!communicate_data ?cdm ?disaster ?area1 ?area2) )
34 )
35 ))
```

[1]The complete HTN and PDDL specifications of the Floods domain can be found at https://github.com/smart-pucrs/floods-domain

The `navigate` operator is the move action for UAVs. It is different from UGVs and USVs because UAVs are not limited to move only through ground or water paths. The remaining operators are pretty straightforward: `take_picture` takes a picture of a flood disaster if that area is visible from where the agent is; and `communicate_data` sends the image of a disaster to a CDM, if it is in range of the area that the agent is located in.

There are two methods for navigation, `navigate` and `navigate_uav`. The latter has two branches: the first checks if the agent is already in the destination, while the second moves to the destination. The `get_picture` method tests if there is evidence of a disaster, and where it can be viewed from. Then it navigates to that area, captures an image of the disaster, and calls the `send_data` method, which moves to an area that is in range of a CDM, and finally transmits the data.

We show an example for a problem specification of an UAV in Listing 3.2. The *:unordered* flag in SHOP2 formalism indicates that there is no particular order for the goals to be achieved. Domain and problem specifications for UGVs and USVs are located in Appendix A.

Listing 3.2 – HTN problem specification of an UAV agent.

```
1  (defproblem uav1 floods-uav (
2        (area area1)
3        (area area2)
4        (area area3)
5        (area area4)
6        (disaster flood1)
7        (disaster flood2)
8        (cdm cdm1)
9        (cdm_at cdm1 area1)
10       (at area1)
11       (visible_from flood1 area1)
12       (visible_from flood2 area4)
13       (in_range area2 area1)
14       (in_range area1 area1)
15 )
16 (:unordered
17       (:task get_picture flood2)
18       (:task get_picture flood1)
19 )
20 )
```

3.2 JaCaMo MAS

Listing 3.3 contains the necessary information for starting the MAS that represents the problem described in Listing 3.2, Listing A.3, and Listing A.4. The *cdm* is represented as an agent, to simulate the input provided by humans in a real world sce-

nario, such as creating new social goals for the robots. This `agent` descriptor contains the name and code filename of the agent, which workspaces they should `join`; and what artifacts they should `focus`.

Listing 3.3 – The jcm file for the Floods MAS.

```
mas floods {
    agent cdm1 : cdm.asl {
        join: env
        focus: env.cdm1
    }
    agent uav1 : uav.asl {
        join: env, robots
        focus: env.area1, env.area2, env.area3, env.area4, env.flood1, env.flood2, env.flood3, env.flood4, env.cdm1, robots.uav1
    }
    agent ugv1 : ugv.asl {
        join: env, robots
        focus: env.area1, env.area2, env.area3, env.area4, env.flood1, env.flood2, env.flood3, env.flood4, env.cdm1, robots.ugv1
    }
    agent usv1 : usv.asl {
        join: env, robots
        focus: env.area1, env.area2, env.area3, env.area4, env.flood1, env.flood2, env.flood3, env.flood4, env.cdm1, robots.usv1
    }
    workspace env {
        artifact area1: floods.Area(["area2"],["area3"])
        artifact area2: floods.Area(["area1,area4"],[""])
        artifact area3: floods.Area([""],["area1"])
        artifact area4: floods.Area(["area2"],[""])
        artifact flood1: floods.Disaster(["area3"])
        artifact flood2: floods.Disaster(["area4"])
        artifact flood3: floods.Disaster(["area3"])
        artifact flood4: floods.Disaster(["area4"])
        artifact cdm1: floods.Cdm("area1",["area2","area1"])
    }
    workspace robots {
        artifact uav1: floods.VehicleUav("area1")
        artifact ugv1: floods.VehicleUgv("area1")
        artifact usv1: floods.VehicleUsv("area1")
    }
    organisation org: floods-os.xml
}
```

The next block from the .jcm file is the workspace definitions. They are followed by the name assigned to the workspace and the definition of all artifacts that are inside this particular workspace. Each artifact has a unique name, and the initial parameters that are assigned to their respective observable properties. For instance, the *Area* artifacts receive two different lists as initial parameters, one list for each of its observable properties: *water path* and *ground path*. For example, *area1* has a water path with *area2* and a ground path with *area3*. Finally, the last block is the definition of the organisation.

In this case, only the filename is informed, but groups and schemes could also have been instantiated.

Jason Agents

In Jason, as in most agent platforms, communication between agents is based on speech-act theory. Formal semantics of speech-act for AgentSpeak can be found in [137]. For example, the *cdm1* agent executes the action:

```
.send(uav1, achieve, get_picture(flood2))
```

to send a message to agent *uav1* with the *achieve* performative, directing it to adopt a new goal and causing a goal addition event `+!get_picture(Disaster)[source(A)]`, where *A* is unified with the name of the message's sender, *cdm1*. Basically, *cdm1* is asking *uav1* to capture an image of the *flood2* event.

Listing 3.4 – UAV agents' code snippet.

```
+!navigate(To)
    : at(From)
<-
    !navigate(From,To).

+!navigate(From,To)
    : at(To).

+!navigate(From,To)
    : .my_name(Name)
<-
    navigate(From,To)[artifact(Name)].

+!take_picture(Area,Disaster)
    : visible_from(Areas)[artifact(Disaster)] & .my_name(Name) & cdm_at(CdmAt)[artifact(Cdm)]
<-
    !navigate(Area);
    take_picture(Area,Disaster,Areas)[artifact(Name)];
    !communicate_data(Cdm,Disaster,Area,CdmAt).

+!communicate_data(Cdm,Disaster,At,To)
    : cdm_at(CdmAt)[artifact(Cdm)] & in_range(Areas)[artifact(Cdm)] & .my_name(Name)
<-
    !navigate(To);
    communicate_data(At,To,CdmAt,Disaster,Areas)[artifact(Name)].
```

We show an excerpt of the Jason agent code for UAVs in Listing 3.4, containing some of the methods found in Listing 3.1. The `!` symbol, inside of a plan body, represents the addition of a goal, which will activate another plan. In the HTN formalism, *!* represent operators (i.e., actions), which in JaCaMo are identified by the suffix containing the artifact name that where that action should be executed.

Differently from SHOP2 methods, Jason allows the use of plans with the same name. There are three different navigate plans, all with the same name but each has different purposes. The Jason interpreter will try each one, in a top-to-bottom order. The first *navigate* plan (line 1) is the same as the first navigate method from line 17 in Listing 3.1. The `.my_name` internal action returns the name of the agent, which is then used to invoke the artifact associated with that agent, whenever an action from the agent's artifact is to be executed. The other two navigate plans (lines 6–12) are the two branches from lines 21–26 in Listing 3.1. Finally, plans for taking a picture and communicating data are also shown. The remaining codes for UGVs and USVs are shown in Appendix A.

CArtAgO Environment

Next, we show the artifact classes that were instantiated in the jcm example from Listing 3.3. Listing 3.5 is the artifact that contains information about flood events. The observable property *visible_from* denotes from which area (or a list of areas) that flood event is visible from.

Listing 3.5 – The disaster artifact.

```
1 public class Disaster extends Artifact {
2     void init(String area) {
3         defineObsProperty("visible_from", area);
4     }
5 }
```

In Listing 3.6, we show the artifact for storing information about areas. The area artifact has two observable properties, *water_path* and *ground_path*, both are able to store string lists containing areas that are connected to an area artifact instance. It also has four operations that are linked to another artifact (to the cdm artifact), to add or remove water and ground paths between two areas.

Listing 3.6 – The area artifact.

```
1 public class Area extends Artifact {
2     void init(String wpath, String gpath) {
3         defineObsProperty("water_path", wpath);
4         defineObsProperty("ground_path", gpath);
5     }
6 }
```

Listing 3.7 has the code for the CDM artifact. This artifact is linked to the four instances of the area artifact (area1, area2, area3, and area4). It has the observable

properties *at_cdm*, that contains the area where the CDM is situated, and *in_range*, that has a list of areas that are in range of the CDM for data transmission. The four operations are used to call upon the linked operations from area artifacts. They represent the ability of the CDM to inform agents of changes caused by flood events, since the vehicles do not possess any advanced sensory capabilities. For example, a flood event can be so disastrous that it can simply wipe out any ground paths between two areas. A flood event could also create water paths that did not previously exist between two areas. Similarly, when a flood recedes, the ground paths that were wiped out can be traversed again, and the water paths that were created would no longer exist.

Listing 3.7 – The cdm artifact.

```
1  public class Cdm extends Artifact {
2      void init(String area) {
3          defineObsProperty("cdm_at",area);
4          defineObsProperty("in_range",range);
5      }
6      @OPERATION void addWaterPathArea(o,a) { execLinkedOp(o,"addWaterPath",a);}
7      @OPERATION void removeWaterPathArea(o,a) { execLinkedOp(o,"removeWaterPath",a);}
8      @OPERATION void addGroundPathArea(o,a) { execLinkedOp(o,"addGroundPath",a);}
9      @OPERATION void removeGroundPathArea(o,a) { execLinkedOp(o,"removeGroundPath",a)
       ;}
10 }
```

The box artifact, Listing 3.8, has the observable property *box_at* defining the current location of the box. This artifact has a linked operation to update the location of the box which can only be executed by an UGV agent's artifact, which are the only ones equipped with the capabilities to move boxes.

Listing 3.8 – The box artifact.

```
1  public class Box extends Artifact {
2      void init(String location) {
3          defineObsProperty("box_at", location);
4      }
5   @LINK
6      void updateLoc(String location){
7         getObsProperty("box_at").updateValue(location);
8      }
9  }
```

In Listing 3.9, we show the artifact for UAV agents. The *at* observable property represents the current location of the agent. Actions are the same from operators in Listing 3.1, with some minor differences to work in JaCaMo's environment representation. Similarly, the artifact for UGV and USV agents are described in Appendix A.

Listing 3.9 – The UAV artifact.

```
1  public class VehicleUsv extends Artifact {
2      void init(String area) { defineObsProperty("at",area); }
3      @OPERATION void navigate(String from, String to) throws InterruptedException {
4          ObsProperty cond1 = getObsProperty("at");
5          if (cond1(from)) {
6              getObsProperty("at").updateValue(to);
7          } else { failed("Action navigate has failed."); }
8      }
9      @OPERATION void take_picture(String area, String disaster, String areas) throws
       InterruptedException {
10         ObsProperty cond1 = getObsProperty("at");
11         if (cond1(area) && areas.contains(area)) {
12             defineObsProperty("have_picture", disaster);
13         } else { failed("Action take_picture has failed."); }
14     }
15     @OPERATION void communicate_data(String at, String to, String cdmAt, String disaster,
       String areas) throws InterruptedException {
16         ObsProperty cond1 = getObsProperty("at");
17         ObsProperty cond2 = getObsPropertyByTemplate("have_picture", disaster);
18         if (cond1(at) && cond2(disaster) && to.equals(cdmAt) && areas.contains(to)) {
19             defineObsProperty("communicated_data", disaster);
20             removeObsPropertyByTemplate("have_picture", disaster);
21         } else { failed("Action communicate_data has failed."); }
22     }
23 }
```

Moise Organisation

Listing 3.10 shows the structural specification for the floods organisation. Five roles are defined, with three of them being specialisations of the *vehicle* role. The group specification defines the cardinality for each role that is required to fully form the group, which reflects the number of agents in the system in our example. Group specifications also contain the designation of links. In this case there are two links, an authority link from the *cdm* to *vehicle* roles, and an acquaintance link from *vehicle* to *vehicle*. The authority link means that the *cdm* can send goals and command directives to vehicles, while the acquaintance link allows vehicles to communicate with each other. In the Floods MAS, the other two Moise specifications (functional and normative, described in Section 2.2.3) are created at runtime by DOMAP, our planning framework.

Listing 3.10 – The structural specification of the floods MAS organisation.

```
<structural-specification>

    <role-definitions>
        <role id="cdm" />
        <role id="vehicle" />
        <role id="uav" >    <extends role="vehicle"/> </role>
        <role id="ugv" >    <extends role="vehicle"/> </role>
        <role id="usv" >    <extends role="vehicle"/> </role>
    </role-definitions>

    <group-specification id="floods">

        <roles>
            <role id="cdm"     min="1" max="1"/>
            <role id="uav"     min="1" max="1"/>
            <role id="ugv"     min="1" max="1"/>
            <role id="usv"     min="1" max="1"/>
        </roles>

        <links>
            <link from="cdm" to="vehicle" type="authority" scope="intra-group"
    extends-subgroups="false" bi-dir="false"/>
            <link from="vehicle" to="vehicle" type="acquaintance" scope="
    intra-group" extends-subgroups="false" bi-dir="false"/>
        </links>

    </group-specification>
</structural-specification>
```

In the next chapter, we introduce the design of DOMAP.

4. DECENTRALISED ONLINE MULTI-AGENT PLANNING

In this chapter, we define the design of our Decentralised Online Multi-Agent Planning framework (DOMAP) [23, 22, 25, 24], a general-purpose domain-independent framework for multi-agent planning in MAS, and describe our implementation of The DOMAP framework in the JaCaMo MAS development platform.

DOMAP consists of several main components: *input language* — a formal description of the language that is used as input for planning and the output plans that are sent for execution; *goal allocation* — a mechanism used to allocate goals to agents; *individual planning* — the planner to be used during each agent's planning phase; and *coordination mechanism* — used during execution to avoid possible conflicts that can occur when agents interact. For each of these components, we describe possible options that could be used and go into detail about the one we incorporated in the framework.

4.1 DOMAP's Design

Multiple agents $(a_1, a_2, ..., a_n)$ interact with an environment to obtain information and carry out their actions. These agents are part of an organisation where they can adopt roles, follow norms, and work towards social plans (structured courses of actions for achieving social goals). In this thesis, we explore the use of coordination at runtime, but show where coordination after or before planning would fit in DOMAP's design, as shown in Figure 4.1. DOMAP is most useful when used in a MAS at runtime that contains the three MAS programming abstractions discussed in Section 2.2.1 — agent, environment, and organisation.

Centralised planners usually assign goals to agents during the search for a solution to a planning problem. In a multi-agent setting, this would constrain the autonomy of the agents, and potentially violate their privacy. Thus, by using task allocation protocols [114], the agents themselves can compete to decide who is better suited to take each goal, then later plan individually (provided coordination mechanisms are in place), giving a higher degree of autonomy and privacy during planning.

Plans that are independently generated by different agents can lead to an infeasible joint solution. According to [148], coordination mechanisms can be viewed as an attempt to achieve a conflict-free joint plan given a set of cooperative or self-interested agents participating in multi-agent planning. Coordination in DOMAP is enforced at runtime when social plans have joint plans involving multiple agents, or plans where an agent's actions can cause conflicts in other agents' plans.

DOMAP is initiated if any of the following situations occur:

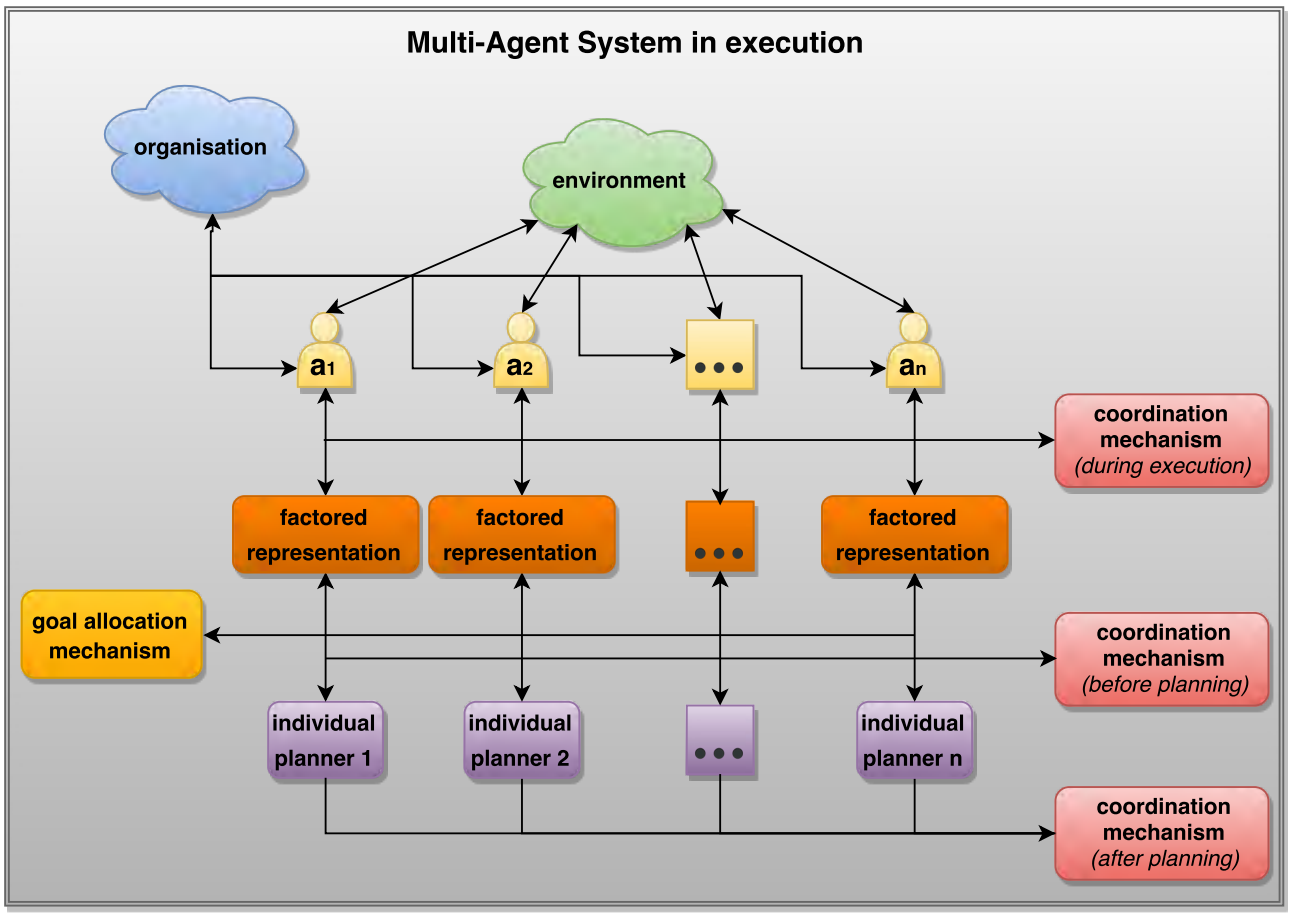

Figure 4.1 – DOMAP design overview.

- New social goals are created for which there are no known social plans.
- The execution of a social plan fails, prompting the drop of all current social goals and intentions related to that social plan. All of the remaining social goals that are still active in that plan are announced as new social goals.

In case of a social plan failure, instead of only replanning for the social goal that has failed, we opted to replan for all social goals related to the plan that has failed. In our experiments, failure would often result from an unforeseen change in the environment, which often have an impact on the whole system, and its effects are not limited to only one goal. Thus, replanning for all of them was more reliable than replanning for only one. However, the performance cost of this choice depends on the domain used and on the effects that these environment changes can cause.

DOMAP can also be used to plan for private goals, which is reduced to single-agent planning and only uses the individual planner component. However, DOMAP does not provide scheduling between private and social goals, so they will be executed in the order that solutions were originally found.

When the MAS is running, if DOMAP is called during any of the situations listed above, the following steps are taken (as summarised in Algorithm 4.1):

1. **Create factored representation interface**: Each agent creates its own representation, containing all the information about the MAS that it has access to at the time. Before starting the planner to search for a solution, the input needs to be translated into a planning formalism that the planner can understand.

2. **Allocate goals**: Agents receive the announcement of social goals that require planning. A mechanism is used to allocate goals to agents that have the best (estimated) chance of finding a potential solution for the goal.

3. **Agents start their individual planner**: Each agent runs a planner to search for a plan to achieve all of the goals it has been allocated. Because planners work individually, and each planner only has access to its own agent's interface, the information that a planner has access to is limited to the information that an agent is allowed to access in that particular moment of the execution, providing some privacy to all agents involved.

4. **Replanning**: If an individual agent's planner fails, the goals it was allocated are re-announced, along with a list of all "*banned*" agents (i.e., agents that have failed to find any solution for those goals), returning to step (2).

5. **Plan execution and coordination**: Agents execute their newly found plans, while following coordination norms.

Algorithm 4.1: DOMAP main function.

```
1 Function plan (Social_Goals)
2     foreach agent ∈ Agents do
3         create_factored_representation_interface;
4     end
5     banned ← ∅;
6     allocate_goals (Social_Goals, banned);
7     individual_planning (Allocated_Goals, banned);
8     execution_and_runtime_coordination;
9 end
```

DOMAP is a general-purpose planning framework, although due to the complexity and variety found in multi-agent planning problems some assumptions are made:

i. **Agents are cooperative**: Agents that are a part of an organisation in the MAS are expected to comply with that organisation's norms and to pursue social goals according to social plans specified in the organisation.

ii. **Planning is deterministic**: Planning state transitions are deterministic, this is a common assumption that is present in many classical planners and traditional HTN planners. However, execution is not necessarily deterministic, the determinism of an

agent's actions in the environment are entirely dependent on how that particular environment was designed in the MAS. If the environment is non-deterministic, it may eventually cause a solution found by DOMAP to fail, at which point agents should be able to automatically recover from it by replanning.

iii. **Planning domains are loosely-coupled**: The number of interactions between agents' actions are low.

iv. **Sufficient social laws are in place**: We assume that the priority law is enough to solve coordination problems in loosely-coupled domains and that conflicting actions were properly identified before execution.

In the following sections, we describe each of the main components of DOMAP represented in Figure 4.1, discussing some of the options found in the literature related to DOMAP's main functionalities, and then detailing the approaches we use and how they were adapted to work in DOMAP.

4.1.1 Input Language

Finding a solution to a planning problem consists of the following process: given a description of the initial state of the world (e.g., agents, environment), a description of desired goals, and a description of possible actions, a planning problem consists of finding a set of plans (i.e., sequence of actions) that when executed from the initial state will lead to the achievement of the desired goals. Therefore, it is beneficial to have a planning formalism in order to formally represent these problems, defining the syntax of the languages that are used for these descriptions. Describing planning domains using a standard formalism promotes greater reuse of research, allowing a fairer comparison between different planners and approaches.

The choice of a planning formalism is usually dependent on which planner is being used. The reason behind this is that most planners have their own formalism, or at least a variation of one that was previously developed and accepted by the planning community.

STRIPS is an early automated planning system from 1971 [53] whose action theory and formalism still provide the basis for many classical planners. However, there were a number of problems with this formulation, such as the difficulty of providing formal semantics for it [89]. PDDL contains STRIPS-like operators, and has been the formalism of choice in several past IPCs. The latest version of PDDL is 3.1[1]. The SHOP2 planner [90] formalism is a well-established and accepted formalism in HTN planning.

[1] https://helios.hud.ac.uk/scommv/IPC-14/repository/kovacs-pddl-3.1-2011.pdf

It differs from PDDL mainly by the use of methods as constraints to prune the search space.

Although it is possible to represent multiple agents using those formalisms, there is no distinction between agents and any other object from the world. This makes it difficult to add important features of MAP, such as privacy, goal allocation, and decentralised planning. Thus, there have been a few extensions of these formalisms to allow for the explicit description of agents.

For example, in MA-STRIPS [17] the authors propose a multi-agent extension of STRIPS formalism for cooperative multi-agent planning. Besides adding the notion of agents containing their own set of actions, dependencies can be identified to classify an agent's action as internal or public.

There is also a multi-agent extension of PDDL 3.1 [71] which is designed to cope with the agents' different abilities and the constructive and destructive nature of concurrent actions. Its design is based on several requirements, such as: modelling concurrent actions with interacting effects; agents can have different actions and goals; straightforward association of agents and actions; distinction between agents and non-agent objects; and inheritance of actions and goals. This extension can also represent problems in temporal, numeric domains.

There are some approaches in MAP that use HTN, such as in [31] and [124], though they do not consider other MAS elements, such as environments and organisations.

This thesis introduces two different input languages that can be used in DOMAP, our novel Multi-Agent Hierarchical Task Network (MA-HTN) and the Factored Representation. We used MA-HTN initially, but we still had to translate a lot of information between the MAS and all of the DOMAP components (goal allocation, individual planner, coordination mechanism). Thus, we designed an interface that collects all of the information available to an agent, which can then be accessed by any DOMAP component without any extra translation. In the experiments shown in this thesis we use factored representation, but we still introduce MA-HTN here since it can be useful in multi-agent HTN planning.

Multi-Agent Hierarchical Task Network (MA-HTN)

Our Multi-Agent Hierarchical Task Network (MA-HTN) formalism is an extension of the centralised single-agent HTN formalism used in SHOP2 planner [90]. MA-HTN can be used to describe multi-agent planning domains and problems. MA-HTN is intended to represent *online* multi-agent planning problems, since domain and problem information are collected at runtime. Thus, unlike in offline planning, where they can be specified before execution (e.g., by a system designer or a computer script), we need

a mechanism to collect all of the necessary data and translate it to an input that is accepted by the planner.

We call this mechanism *translator*, and agents use it to translate their information about the world into domain and problem specifications that can then be passed on to their own individual planner. The translator obtains information about the current state of the world from the MAS in execution, using it to generate the problem representation. The domain representation is generated from the possible actions and plans that agents can use. Each agent has their own problem and domain specification. This provides a decent level of privacy on its own, since each individual planner will only have access to their respective agent domain and problem specifications.

Actions (translated to operators in HTN) from other agents can cause conflicts, either at the moment that action is executed (e.g., concurrent actions) or in the future (e.g., durative actions). For this reason, MA-HTN supports the description of actions that can cause *conflicts*. The recognition of conflicting actions is not automated and should be pre-identified as such before execution, though an external mechanism for that purpose [127] could be used.

Likewise, dependencies between actions can also exist, either as a concurrent action that requires another agent, or actions that depend on actions of other agents to happen first. Similarly to conflicts, MA-HTN also supports the use of *dependency* blocks to identify actions that depend on actions from other agents. These dependency relations also have to be annotated by the MAS developer, so that the translator can automatically add them to the specification.

The notation usually preferred to specify context-free grammars for planning formalisms is the Backus–Naur Form (BNF) [3]. We use BNF grammars to define the specifications of MA-HTN domain and problem representations. A BNF specification is a set of derivation rules, `<symbol> ::= __expression__`, where `<symbol>` is a non-terminal symbol, and `__expression__` is one or more sequences of non-terminal symbols and/or terminal symbols. A symbol is considered terminal, *iff* it never appears on the left side of a rule. Symbols that appear on the left side of a rule are non-terminal symbols, and enclosed between a pair of <>. The `|` symbol represents a choice, for example, if `__expression1__ | __expression2__` appears on the right side of a rule, it means that either `__expression1__` *or* `__expression2__` has to be chosen.

We provide a simplified grammar to improve readability, where each single quote pair that encloses a symbol is considered a string that is expected by the planner. Symbols enclosed by brackets are optional, and symbols preceded by *$* are variables obtained from the MAS and represent terminal symbols. Symbols that end with the * signal, represent that zero or more instances are possible. Symbols that end with the $^+$ signal, represent that one or more instances are possible. Because MA-HTN is an extension of the SHOP2 HTN formalism [90], the language itself is LISP-like, though we

omitted many of the necessary parenthesis that the planner would expect in order to improve readability.

In Listing 4.1, we show our simplified BNF grammar for multi-agent HTN domains. The *$domain-name* variable is defined dynamically by the agent during execution of the MAS, and since there could be multiple calls to the same domain and the specification can be different from the previous call (e.g., a method could be deleted, added, or modified), agents use a counter ID that increments each time planning is invoked. The name of the agent, *$agent-name* is included in the specification to represent which agent this domain belongs to. The *conflict-list* and *dependency-list* are added. Conflicts represent actions that can cause negative interactions between agents, while dependencies are actions that require actions from other agents to succeed. The rest of the definitions are similar to SHOP2 HTN, *operators* are *primitive-tasks*, and *methods* are *non-primitive-tasks* that can eventually be decomposed into *operators*.

Listing 4.1 – MA-HTN BNF grammar for representing domains.

```
1  def-domain              ::= 'defdomain' $domain-name ;
2  agent                   ::= 'agent' $agent-name ;
3
4  task                    ::= primitive-task | non-primitive-task ;
5  primitive-task          ::= '!' $primitive-task-name '?'$variable* ;
6  non-primitive-task      ::= $non-primitive-task-name '?'$variable* ;
7
8  def-operator            ::= ':operator' primitive-task pre-list del-list add-list con-list dep-list ;
9  pre-list                ::= precondition* ;
10 precondition            ::= ('not' $precond-name '?'$variable*) | ($precond-name '?'$variable*) ;
11 del-list                ::= delete* ;
12 delete                  ::= $delete-name '?'$variable* ;
13 add-list                ::= add* ;
14 add                     ::= $add-name '?'$variable* ;
15 con-list                ::= conflict* ;
16 conflict                ::= $action-name ;
17 dep-list                ::= dependency* ;
18 dependency              ::= $action-name ;
19
20 def-method              ::= ':method' non-primitive-task (pre-list task-list+) ;
21 task-list               ::= task+ ;
```

Listing 4.2 shows our simplified BNF grammar for multi-agent HTN problems. Similarly to the domain grammar, the *$problem-name* is specified, along with a reference to its respective *$domain-name*. We also have the explicit agent symbol on line 2. *Facts* can be used to establish types and characteristics of things in the world, for example, *location kitchen* establishes that *kitchen* is a *location* in the world. While *initial-states* are used to represent what is true in the world at that specific moment in the execution, for example, *kitchen dusty* represents that the *kitchen* is *dusty*. *Goals* can be listed as either *ordered* — when the order in which the goals have to be achieved needs

Listing 4.2 – MA-HTN BNF grammar for representing problems.

```
1  def-problem          ::= 'defproblem' $problem-name $domain-name ;
2  agent                ::= 'agent' $agent-name ;
3
4  def-facts            ::= fact-list ;
5  fact-list            ::= fact* ;
6  fact                 ::= $fact-name $fact-parameter+ ;
7
8  def-initial-states   ::= initial-state-list ;
9  initial-state-list   ::= initial-state+ ;
10 initial-state        ::= $initial-state-name $initial-state-parameter+ ;
11
12 def-goals            ::= ( ':ordered' | ':unordered') goal-list ;
13 goal-list            ::= goal+ ;
14 goal                 ::= $goal-name $goal-parameter+ ;
```

to be strictly followed; or *unordered* — when order is not important and the planner is free to find any order between goals.

Factored Representation

Planning input (i.e., domain and problem representation) and output (i.e., the solution) in DOMAP are regulated by a *factored representation* of each agent's knowledge. Each agent's factored representation serves as an interface between the MAS and DOMAP. This interface groups all of the information that can be relevant to DOMAP. Before planning starts, each agent calls its interface, generating up-to-date planning input that contains all the information it has access to. A translator between this factored representation and the input language used as the individual planner has to be provided. For example, if we were using a classical planner, we would need a translator that can parse from our factored representation to PDDL domain and problem files.

We use a translator from DOMAP's factored representation to the HTN formalism. This is pretty straightforward and follows much of the formal work already described in [109], plus a few additions specific to DOMAP shown in Table 4.1. In summary, plans are translated into non-primitive tasks, actions become primitive tasks, beliefs are turned into initial states, and social goals are parsed as the initial task list. Because individual planning starts after goal allocation, the list of social goals for each agent has already been updated to contain only the goals that have been allocated to them. The solution, a sequence of actions, found by each individual planner is then translated back into plans.

We assume that there are mechanisms in place, provided either by the MAS developer or the development platform, that ensure that an agent cannot access information that it is not privy to. We do not impose any restrictions in agent communication, agents can freely exchange information via message passing, if they want to. Thus, al-

Table 4.1 – Factored representation and HTN formalism equivalences.

factored representation	**HTN formalism**
belief	state
plan	non-primitive task
plan's preconditions	non-primitive task constraints
action	primitive task
action's preconditions	primitive task's precondition list
action's effects	primitive task's add and delete list
social goals	initial task network

though we do not enforce privacy, our agents also do not violate it unless specifically programmed to do so. If the system requires any level of privacy, our factored representation interfaces can serve as a foundation, assuming each agent is restricted to its own interface.

4.1.2 Goal Allocation

Task/goal allocation mechanisms and protocols have been extensively researched, many options can be found in the literature. A Vickrey [136] auction is an example of an auction protocol that is frequently used in MAS. In this protocol, each agent can make one closed bid (i.e., other agents in the system do not have access to this value) for a particular goal. The task of achieving that goal is assigned to the highest bidder for the price of the second-highest bidder. This means that each bidding agent should bid their true values (i.e., exactly what they think it's worth to them), since this will provide them with the best chance of winning without overspending. An example of its use in MAP can be found in [78], and it is discussed in Section 2.3.

Market simulations and economics can also be used to allocate large quantities of resources to agents [141]. For example, in [139], a decentralised market protocol for allocating tasks and scarce resources among agents is presented. The authors of [30], suggest how costs and money can be turned into a coordination device. In the work of [140], a market-oriented programming approach to distributed problem solving is proposed, along with an algorithm for distributed computation of competitive equilibria. In the context of self-interested agents and value-oriented environments, there is also the work done in [107], where agents reason about the cost of their decision making in the dispatch and routing of vehicles.

Contract Net Protocol (CNP) [120] is also frequently used both in MAS and MAP. According to Smith's original concept from his work in 1980 [120], the contract net protocol "has been developed to specify problem-solving communication and control for nodes in a distributed problem solver". In this protocol nodes enter a negotiation

process involving auctions, where the winner is awarded the task. Nodes in the CNP can assume one of two possible roles, initiator (manager) or bidder (contractor). The initiator is responsible for announcing new tasks in the CNP (i.e., create new auctions), monitoring its activity, awarding tasks to winners, and processing the results of the task's execution. Bidders decide which auctions to take part in, calculate their bids, and execute the tasks that they are awarded.

The original protocol contained the following stages:

1. **task announcement**: The initiator starts a contract negotiation by contacting bidders and sending them the announcement specification. There are four main elements in the announcement specification:
   - **eligibility**: list of criteria that a bidder must meet to be eligible to submit a bid;
   - **task abstraction**: a description of the task;
   - **bid description**: the expected format of a bid;
   - **expiration time**: a deadline for receiving bids.
2. **Announcement processing**: Bidders process the announcement and check if they are eligible to bid.
3. **Bidding**: Bidder nodes decide on their bid, according to the bid description of the task.
4. **Bid processing**: The initiator awards the contract to the best bid, either after the deadline of the auction, or if it finds any of the bids to be satisfactory.
5. **Termination**: Bidders that were awarded a task have to report about the task conclusion back to initiators. An initiator can terminate a contract at any moment, by sending a termination message.

Using Contract Net Protocol as a goal allocation mechanism

We use a variation [22] of the original CNP design [120] for goal allocation in DOMAP. The organisation acts as the initiator, responsible for starting new contracts for social goals that have not been previously assigned to any agent. Agents from that organisation can then place bids for each contract. Some modifications were also made on the goal announcement, bid, and award specifications. Notably, some unnecessary elements have been removed. For example, there is no need to describe how a bid should be specified, since this information is strictly domain-dependent.

In Listing 4.3, we show an overview of the specification for a social goal announcement. The *to* field allows the announcement to be sent either to all agents in the

organisation (broadcast), or to a specific group of agents within the organisation (multicast). Each announcement is identified through a unique *ID*. The *goal* contains both the name of the goal and the goal specification, such as preconditions for example. The *eligibility* field can be used for restricting goals to certain roles, and/or to agents that have specific plans in their plan library. Finally, the *deadline* limits the time, in milliseconds, that the initiator will accept bids for the social goal.

Listing 4.3 – Goal announcement specification.

```
1  to          ::= all | group_id ;
2  from        ::= organisation ;
3  id          ::= announcement_id ;
4  goal        ::= goal−name, goal−spec ;
5   eligibility ::= role &/| plans ;
6  deadline    ::= time (in milliseconds) ;
```

A high-level description of DOMAP's goal allocation is shown in Algorithm 4.2. Agents place a multi-valued bid, a 5-tuple containing the following criteria: recursion (0 or 1), total number of actions expanded, total number of plans expanded, and maximum tree depth and width found while expanding their goal-plan tree. The "*recursion*" criteria indicates if any recursion was present in the agent's plan library, that is, whether any recursive plan was expanded during the bid calculation for a social goal. If an agent did not expand any recursive plan, then this indicates that the agent may potentially solve the social goal in less steps than agents with recursive plans. These criteria are used as heuristics by the organisation to allocate social goals to agents with the best (according to the heuristic chosen for that application) bids.

Algorithm 4.2: DOMAP goal allocation.

```
1  Function allocate_goals (Social_Goals, banned)
2      announce (Social_Goals, banned);
3      foreach agent ∈ Agents do
4          foreach socialgoal ∈ Social_Goals do
5              if (agent, socialgoal) ∉ banned then
6                  bid (agent, socialgoal, bid);
7              end
8          end
9      end
10     award (Social_Goals, bids);
11 end
```

Although domain-dependent algorithms for determining an agent's bid will often provide better results, we provide a domain-independent function that agents can use to calculate their bid (line 6 in Algorithm 4.2). This function performs a breadth-first expansion of a social goal using the agent's plan library, essentially exploring structures similar to goal-plan trees in the process. Goal-plan trees [127] are tree structures of goals

whose children are the plans that achieve it, and the children of a plan are subgoals. For the purpose of comparing goal-plan trees between agents as used in our approach, we ignore goals and subgoals from the tree. We also opted for not saving the whole goal-plan tree, instead, we update the measurements that will be part of the bid, and save only the plans that need to be immediately expanded. This makes the expansion process faster and uses less memory.

When comparing plan trees between agents, we can classify them according to their topology into four different types: *recursive distinct plan tree*, when a recursive plan was used and plan trees are sufficiently distinct from each other; *non-recursive distinct plan tree*, when no recursive plan was used and plan trees are sufficiently distinct from each other; *recursive similar plan tree*, when a recursive plan was used and plan trees are similar or identical to each other; and *non-recursive similar plan tree*, when no recursive plan was used and plan trees are similar or identical to each other.

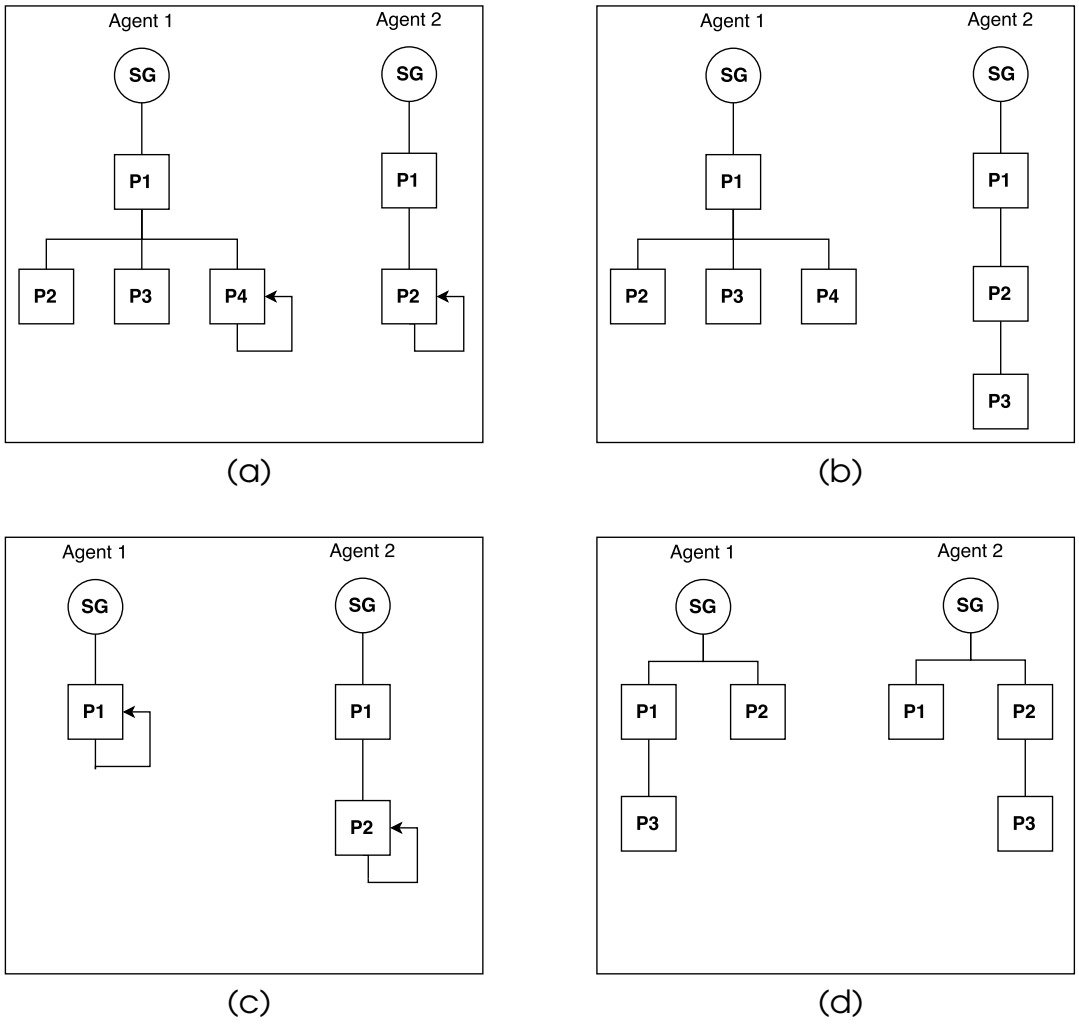

Figure 4.2 – Possible plan trees: (a) recursive distinct plan tree; (b) non-recursive distinct plan tree; (c) recursive similar plan tree; (d) non-recursive similar plan tree.

In Figure 4.2, we can observe some examples for all four plan tree topologies. Figure 4.2(a) shows the recursive distinct plan tree comparison, where plan trees contain recursive plans and the trees are different from each other, which is evident by comparing the maximum width of both trees: 3 for *agent1* and 1 for *agent2*. Figure 4.2(b) shows the non-recursive distinct plan tree, where plan trees do not contain any recursive plans and the trees are once again different from each other. Figure 4.2(c) shows the recursive similar plan tree, where plan trees contain recursive plans and the trees are very similar to one another (same maximum width). Figure 4.2(d) shows the non-recursive similar plan tree, where plan trees do not contain any recursive plans and the trees are again very similar.

We do not show the variations when some plan trees contain recursive plans and some do not, because if there is at least one agent that has a plan tree with recursive plans, then it can be considered to belong in the *recursive distinct plan tree* type.

During the expansion of a social goal, our algorithm ignores the preconditions of plans, that is, we do not apply an action theory. This would involve performing lookahead, which is essentially planning, and would have a high computational cost. Instead, we opted for a quick expansion of the plan library, selecting all plans that are related to the social goal, while ignoring the context of these plans. This relaxation allows for quick computation of the statistics for the goal allocation phase, sacrificing allocation quality for faster allocation times.

AgentSpeak-like plan libraries often contain several recursive plans, which leads to infinite expansion of goal-plan trees [127]. Agents with recursive plans can get stuck in a loop during execution, either because actions on real-world applications are non-deterministic, or because the MAS programmer made a mistake. Although we considered limiting or ignoring these recursive plans, we found out from our experiments that this can be a valuable information to have, and the disadvantage of infinite loops are surpassed by using deadlines.

Contract nets have a deadline in order to prevent initiators to wait indefinitely for bids. We use it to stop the infinite expansion that can happen when agents get stuck in a loop, or when agents have large plan libraries. Agents expand the social goal up until they are close to the deadline, at which point they stop the expansion and use measurements of the expanded tree to form their bid.

While testing breadth-first against depth-first, we found that breadth-first provided better results, especially when dealing with plan libraries containing recursive plans. Breadth-first avoids getting stuck in an infinite loop by expanding only recursive plans. Instead, it tries to expand other available plans, providing better information and more coverage of the plan library. Our algorithm for domain-independent bid calculation is shown in Algorithm 4.3.

Algorithm 4.3: Breadth-first expansion of a social goal.

```
1  Function expand(goal, deadline)
2      Plans ← relevant_plans(goal);
3      if Plans = ∅ then
4          return "not eligible";
5      end
6      rec, n_actions, n_plans, m_depth ← 0;
7      m_width ← |Plans|;
8      Subplans ← ∅;
9      while there exists {plan} ∈ Plans do
10         if time() ≥ deadline then
11             return (rec, n_actions, n_plans, m_depth, m_width);
12         end
13         if {plan} is recursive then
14             rec ← 1;
15         end
16         n_actions ← n_actions + count_actions(plan);
17         n_plans ← n_plans + 1;
18         Goals ← goals(plan);
19         Subplans ← Subplans ∪ relevant_plans(Goals);
20         Plans ← Plans \ {plan};
21         if Plans = ∅ then
22             if |Subplans| > m_width then
23                 m_width ← |Subplans|;
24             end
25             m_depth ← m_depth + 1;
26             Plans ← Subplans;
27             Subplans ← ∅;
28         end
29     end
30     return (rec, n_actions, n_plans, m_depth, m_width);
31 end
```

The expansion starts with the social goal at the root of the tree. The agent uses a $relevant\_plans$ function that returns all plans in the agent's plan library that can be used to achieve that particular goal. If the set of such $Plans$ is empty, then the agent is not eligible to bid for this contract. Otherwise, the information used for determining the agent's bid is initialised: rec is the presence of recursion; $n\_actions$ is the total number of actions found in all of the plans that were expanded; n_plans is the total number of plans that were expanded; m_depth is the maximum depth of the tree; and m_width is the maximum width of the tree, which initially receives the cardinality of the $Plans$ set, indicating the initial width of the tree. The $Subplans$ set is initially empty. Parameter $deadline$ is the maximum time after the start of expansion that the algorithm can run for. It is expected that when calling the $expand$ function, the deadline given for the algorithm is slightly lower than the CNP deadline, so that the agent has time to communicate its bid to the organisation.

At the start of a plan's expansion, the agent uses a function $time()$ to get the time that has passed since the start of the $expand$ function, and checks if that value is greater than or equal to $deadline$. If it is, then the algorithm stops the expansion and returns the bid measurements found up to that moment.

While there remains any $plan$ in the $Plans$ set, we check if the plan contains recursion, increase the counter of total plans expanded, add the number of actions found in the body of the plan to the total number of actions, and assign all the subgoals found in the body of the $plan$ to the $Goals$ set. Then, the agent uses once again the $relevant\_plans$ function to get all relevant plans, but now for each of the subgoals in the $Goals$ set. The $plan$ that was expanded is removed from the $Plans$ set. If this was the last plan and the $Plans$ set is now empty, then the agent checks if the cardinality of the $Subplans$ set is higher than our current m_width, in which case the cardinality of the $Subplans$ set becomes the current maximum width of the tree. After that, we increase the maximum depth counter, and move the $Subplans$ set to the $Plans$ set (this effectively give us breadth-first search, but doing it in this particular way allows us to take all bid measurements that we need).

When both sets are empty, or the deadline is past, the algorithm returns the multi-valued bid with relevant information collected during the expansion of the goal-plan tree for the given social goal.

We also allow concurrent contract net announcement and bidding. This, in turn, can cause an agent to win all social goals. To avoid that, we added priorities to the bid selection heuristics used by initiators. The award function from line 10 in Algorithm 4.2 works as follows: after bidding has finished for all active contract nets, the initiator first prioritises processing the results of contracts that had agents not eligible to accomplish the task, since contracts that had valid bids from all agents are easier to allocate, thus, possibly increasing fairness and resulting in a more even distribution of goals between all agents. The first bidding attribute to be processed is *recursion*, giving priority to agents that did not expand any recursive plans. If there are any such agents, then the contract is awarded to the agent with the lowest *total number of actions* among those that did not use recursion. Otherwise, the contract goes to the agent with the higher *maximum tree width*. In both cases agents that have not been awarded any social goal yet are prioritised, and in case of any ties, the first bid that was processed wins.

Agents with higher total number of expanded plans represent agents with the best computing power available (assuming that they expanded plans up until the deadline). A higher number of actions represent agents that may require longer steps in order to solve a social goal. When compared to other agents, agents with a lower maximum depth often represent that there were no expansion of recursive plans. Agents with a higher maximum depth can represent either very linear plan trees (few options), or the

presence of recursive plans. Higher maximum width often indicates that the agent has more options available to accomplish the social goal.

After all social goals have been allocated, that is, each social goal has a contractor agent in charge of achieving it, then the next phase of DOMAP can start. We assume here that every goal will eventually be allocated, meaning that there is at least one eligible agent for each social goal. Allocating a goal to an agent does not guarantee that the agent will be able to find a solution (this depends on the results of the planning phase) and execute it successfully (which depends on the execution phase). Instead, goals are allocated to agents that have shown a better chance of doing so according to bid selection heuristics.

An interesting property of CNP, is that it allows for bidders to partition the task that was awarded to them into subtasks, which can then be announced as new contracts. In our approach, it could be interesting to allow agents to become initiators of tasks that they believe they can only do a part of the task, subcontracting the rest. This would be especially useful for solving social goals in tightly-coupled domains, i.e., where actions required to achieve the goals have many dependencies.

Furthermore, the authors of [147] propose some interesting extensions that could provide several benefits if integrated with our CNP mechanism. Two main ideas are discussed. First, the authors suggest adding a threshold to limit the number of contracts that each bidder can participate, which could be useful in domains with many social goals where agents can only (practically) pursue some of these goals. And second, a degree of availability is suggested to allow the initiator to also consider availability of the bidders, along with their respective bid.

4.1.3 Individual Planning

Planning has already been widely covered in Section 2.1. We use an HTN planner for the individual planning phase of DOMAP. In a HTN planner, planning occurs by using methods to decompose tasks recursively into smaller subtasks. For each non-primitive task, the planner chooses an applicable method, uses it to decompose the task into subtasks, and then chooses other methods to decompose the remaining subtasks, until it reaches primitive tasks that can be performed using operators [90]. If no solution is found, then the planner backtracks and try other methods.

Anytime planning algorithms can be interrupted and resumed with little overhead, can provide increasingly good answers over a range of response times [44]. DOMAP requires a similar, but simpler mechanism, that is able to stop the search for a solution if it takes longer than a time limit, and return the best plan found up to that point.

Although we do not cover non-deterministic planning domains in this thesis (Assumption *ii* from Section 4.1), we refer the interested reader to an extension of SHOP2 to support non-deterministic domains that was proposed in [73]. Their algorithm combine the search control of HTN planning with the state abstraction of symbolic model-checking based on binary decision diagrams. Their argument is that by using symbolic model-checking they can exploit search control heuristics for pruning the search space. And using HTN planning allows them to use propositional formulas for a compact representation of states and transformations over such formulas for efficient exploration in the search space.

As for supporting probabilistic domains, in [72], the authors describe adaptations of SHOP2 to work in probabilistic domains such as the ones represented by MDPs. The authors achieve that by describing how to include the search control algorithms from SHOP2 into any forward-chaining MDP planner. Their experiments show that this results in better performance for MDP planning algorithms.

Using SHOP2 as the individual planner

HTN planning and SHOP2 [90] were already discussed in Section 2.1.2, though in this section we provide more details about our use of SHOP2 in DOMAP. No modifications were made to the actual planning algorithm and search techniques of SHOP2, only its interaction with the other DOMAP components.

SHOP2 has several search algorithms that can be set with the following parameters:

- **first**: depth-first search that stops at the first plan found.
- **shallowest**: depth-first search for the shallowest plan, or the first such plan if there are more than one.
- **id-first**: iterative-deepening search that stops at the first plan found.

There is also an *all* option for each of these values, which searches for all possible plans. These usually take a long time to finish, and thus, given the nature of online planning, are not very relevant to DOMAP. The default value for the search used in SHOP2 within DOMAP is *shallowest*, since it is the most useful when used with the time-limit in planning that DOMAP requires. The best search algorithm will depend on the domain used, and may require experimentation to discover.

The next important parameter for DOMAP in SHOP2 is the *time-limit*, which can be either *nil* or a number. If *nil*, then no time limit is imposed on planning. If the argument is a number, SHOP2 checks the elapsed time at the start of each planning step, and if the number of seconds elapsed is greater than the argument value, SHOP2

immediately terminates and returns the plan with the lowest cost found within the given time limit.

The high-level function for individual planning is shown in Algorithm 4.4. DOMAP also has a replanning mechanism. If the individual planning of one (or more) allocated social goal(s) fail, the agent sends the respective social goal(s) back to the organisation. When all agents finish their individual planning, the organisation will either start a new round of goal allocation and individual planning (if it received at least one social goal back), or allow agents to start their execution (if no social goals were received back as having failed individual planning). The name of agents who have failed planning are added to a *banned* list, which is sent together with the social goal when it is announced again in a new round. The number of replanning rounds will depend on the efficiency of the bid selection heuristics that are being used, and how restrictive the domain's goals are (e.g., goals that can only be completed by a select handful of agents can cause a negative impact on performance).

Algorithm 4.4: DOMAP individual planning.

```
1  Function individual_planning (Allocated_Goals, banned)
2      foreach agent ∈ Agents do
3          start_HTN_planner (agent, Allocated_Goals);
4          if planning_failed (agent, Allocated_Goals) then
5              foreach socialgoal ∈ Allocated_Goals do
6                  banned ← banned ∪ {(agent, socialgoal)};
7              end
8          end
9      end
10     if banned ≠ ∅ then
11         allocate_goals (Allocated_Goals, banned);
12     end
13 end
```

Alternatively, our replanning mechanism can also take advantage of roles defined in the organisation, by supporting the banning of roles instead of agents; that is, all agents that have the same role as the agent who failed planning for a goal will be considered ineligible to bid for that goal in any subsequent round. This is useful if there is some prior knowledge about the domain that indicates that if an agent fails planning for a social goal, then any other agent of the same role will also fail. If this is not the case, banning of roles should not be used, as it can cause DOMAP to no longer be able to find a solution. The advantage of using roles is that the worst case in goal allocation is $R - 1$, with R the number of roles, which is typically much smaller than the number of agents. The worst case of banning agents instead of roles in goal allocation is $A - 1$ rounds, with A being the number of agents, but it has the advantage that it is complete (if there is a solution, it will eventually find one).

In the experiments reported in this thesis we obtained similar results for both approaches in the Floods domain, but banning roles was not useful in either Rovers or Petrobras domain. Thus, our DOMAP results from Chapter 6 use agent bans.

Execution of the solution starts only when all agents who were awarded social goals finish their individual planning. This includes any replanning rounds that might be necessary.

4.1.4 Coordination Mechanism

As established by Cox and Durfee in [34], "agents that share an environment and that want to achieve collective (as opposed to selfish) goals need to coordinate their planned actions at least to avoid interfering with each other, and preferably to help each other". They also establish a particular class of coordination problems, which they call the multi-agent plan coordination problem. Plans in this classification are represented by partial-order causal-link plans, in order to capture temporal and causal relations between steps. There are some limitations to their classifications though, as some coordination problems cannot be modelled, such as problems where agents would have to reallocate their activities, or problems where additional action choices are available if agents work together.

There are coordination mechanisms that focus on how to coordinate agents before they even start creating their plans, by defining rules or constraints for agents to prevent them to produce conflicting plans [**?**]. Below we describe some techniques that can be used when coordination is performed before planning, namely *temporal decoupling* and *summary information*.

Scheduling problems can often be found as a subset of multi-agent planning problems, and require specific coordination techniques aimed at solving them. Temporal decoupling can be used to efficiently find and represent the complete set of consistent joint schedules in a decentralised and privacy-maintaining manner. When dealing with large problems, summary information can be annotated into each abstract operator about all of its potential needs and effects, such as conflicts. Approaches for both temporal decoupling and summary information have already been discussed in Section 2.3.

Coordination mechanisms that are focused on how to coordinate agents after planning, aim at the construction of a joint plan, given the individual subplans of each of the participating agents [43]. Below we describe some techniques that can be used when coordinating after planning, namely *plan merging* and *plan repair*.

Plan merging techniques, as described in [149, 55], usually involve generating plans for each goal individually, ignoring how the solution might affect other goals, and then merge together all solutions, handling the interactions that appear during this

process. The idea behind multi-agent plan merging, as described in [42], is that each agent creates a plan for its own goals, then these plans are analysed to detect and resolve conflicts, and possibly to exploit positive interactions.

Plan repair can be seen as planning with the re-use of fragments of the old plan, and can be used to effectively simplify coordination. The authors of [70] argue that in decentralised systems where coordination is required to achieve joint objectives, attempts to repair failed multi-agent plans should lead to lower communication overhead than re-planning from scratch. Thus, this coordination mechanism is especially useful in domains that require coordination with limited to no communication between agents.

Coordination techniques in MAS may also be broadly categorised by whether they are online or offline. Online techniques aim to equip agents with the ability to dynamically coordinate their activities, for example by explicitly reasoning about coordination at runtime. In contrast, offline techniques aim at developing a coordination mechanism at design time. Online is potentially more flexible, and may be more robust against unanticipated events. Offline benefits from reasoning about coordination before execution, thus reducing the decision-making time that agents would have to spend in runtime [144].

In systems with multiple agents, certain rules (e.g., laws, protocols, norms) can be imposed upon the agents in order to regulate their interactions and communication, so as to enable agents to achieve the desirable goals of the system [58]. When these conventions are widespread, they are called social laws. According to [62], "a social law can be understood as a set of rules imposed upon a multi-agent system with the goal of ensuring that some desirable global (coordination) behaviour will result". These laws will on the one hand constrain the plans available to the agents, but on the other hand will guarantee certain behaviours.

Our coordination mechanism is online, that is, coordination is done at runtime of the MAS via social laws.

Using social laws to coordinate agents at runtime

The model of social laws that we based our coordination mechanism for DOMAP is the one established by Shoham and Tennenholtz in [116, 117]. In that model, social laws were used to restrict the activities of agents so as to ensure that all individual agents are able to accomplish their personal goals. We follow a similar idea, although agents here aim to achieve social goals, and thus, are compelled when joining the organisation to follow these social laws.

Social laws can coordinate agents by placing restrictions on the activities of the agents within the system. One of the advantages of using social laws is that agents can solve some coordination problems without having to communicate directly with each

other. The purpose of these restrictions are twofold: they can be used to prevent some destructive interaction from taking place (conflict); or they can be used to facilitate some constructive interaction (dependency).

Shoham and Tennenholtz model of social law is still adopted as the basis for many studies that seek to further extend it with additional capabilities. For instance, in [62], the model was generalised to allow for the objective of a social law (i.e, what the designer intends to accomplish with the social law) to be specified as a logical formula. And in [32], a model for non-deterministic social laws is proposed, applicable to a set of probability distributions that describe the expected behaviors of agents in the system.

Thus, we formally define social laws in our model as:

Definition 3.1. Given a set of agents Ag, a set of actions Ac, a set of states S, a set of preconditions P, and a behaviour β, a *social law* is a tuple (ag,ac,s,P,β) where $ag \in Ag$, $ac \in Ac$, and $s \in S$.

A social law sl constrains a specific action ac of agent ag, considered to be a possible point of conflict (as established in the action description), when the state s satisfies each precondition $\rho_i \in P$. The agent ag then has to follow the behaviour described by β.

As a practical example on the use of social laws, consider a domain consisting of a grid traversed by a group of mobile robots in which agents are responsible for their high-level control. A social law for this domain might institute traffic regulations to insure that agents never collide or get stuck, allowing them to reach whichever nodes they needed.

Figure 4.3 contains an instance of such possible conflict. At each time-step an agent can move to a neighbouring node or stay in the same place. The robots have limited sensory capabilities, meaning that they can only detect other robots located in their immediate proximity (one step away). This means that in three time steps, agents 1 and 2 will collide at coordinate (3,4) if they follow their designated paths. A simple social law that would solve this conflict is to allow only one robot to move in the third time step, while the other holds its action for the next time step.

We call this type of social law, the *priority law*. This is a default social law that DOMAP uses and that can work on many loosely-coupled domains. The behaviour β in the priority law is the following: when two or more agents are trying to execute conflicting actions in the same time step, one of the agents is arbitrarily chosen to continue its execution, while all remaining involved agents hold their actions (*null* action, e.g., do nothing) until the next time step. This process continues until there are no longer any conflicting actions trying to be executed in the same time step.

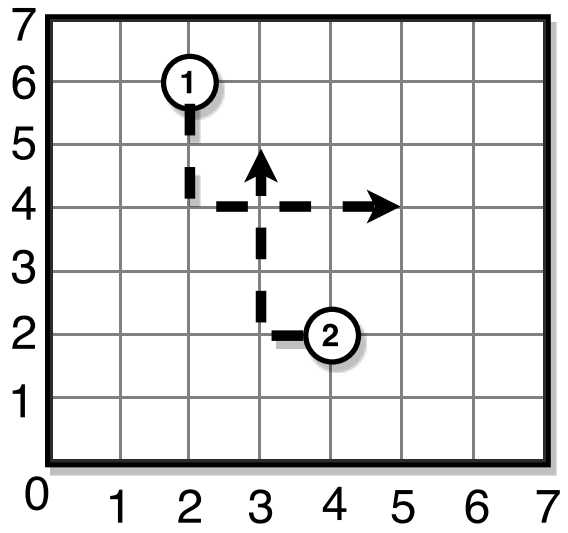

Figure 4.3 – An 8x8 node grid with two agents and their respective paths.

An alternative to alleviating Assumption iv., presented in Section 4.1, is the use of an external mechanism for social law synthesis. Several studies have been conducted on social law synthesis by the community. For example, in [54], an automatic method for the synthesis of minimal social laws in restricted settings is described, where the algorithm starts from a useful social law and decrements the set of constraints until arriving in a minimal form. There is also a social law framework that uses alternating-time temporal logic to synthesise new social laws in [62], by posing it as an alternating-time temporal logic model checking problem.

In this chapter, we discussed the design overview of DOMAP, as well as described each of the main components of DOMAP. In the next chapter, we provide implementation details for each of these components, and describe how they interact with the MAS.

5. DOMAP'S IMPLEMENTATION

This chapter describes the implementation of DOMAP[1] and each of its mains components. DOMAP is designed for online systems, thus applying the use of planning techniques whilst the MAS is running. We use the MAS development platform *JaCaMo*[2] [13] in order to implement and evaluate DOMAP. It combines three different technologies (*Jason*, *CArtAgO*, and *Moise*) into programming abstractions that we found to be a good match for implementing DOMAP — *agent*, *environment*, and *organisation* abstractions. Agents in *Jason* follow the BDI-model and react to events in the system by executing actions on the environment, according to plans available in each agent's plan library. *CArtAgO* uses artifacts to represent the environment, storing information about it as observable properties and providing actions that can be executed through operations. *Moise* adds organisation elements to the MAS such as roles, groups, social schemes (i.e., social plans), and norms.

5.1 Overview

To illustrate the runtime of DOMAP in JaCaMo, consider the overview provided in Figure 5.1. When new social goals emerge, phase 1 activates the contract net protocol mechanism to allocate social goals to agents. In phase 2, the factored representation (containing the agent's knowledge about the world) is translated into the SHOP2 syntax, and used as input for the individual planner (phase 3). If any replanning is necessary, that is, if an agent was not able to find a solution to an allocated goal, the process returns to phase 1 to reallocate any remaining social goals that have failed. After all social goals have been allocated and their plans found, phase 4 starts, where the solution found by each agent's planner goes through an inverse translator, parsing it into plans, and adding them to their respective agent's plan library. Finally, the solution is executed by the agents, in accordance with the social laws (phase 5) embedded into the coordination artifacts.

The CArtAgO infrastructure artifacts (e.g., workspace management) are omitted from the figure. To experiment with privacy levels and to facilitate the translation of the agents' knowledge, we added one artifact per agent (`Artifact` a_1 `...` a_n) that would contain the information and actions that only its associated agent would have. Note that this is not how MAS are traditionally programmed in JaCaMo. These information and actions would either be directly represented within the agent's code or in a pub-

[1] DOMAP's source code can be found at https://github.com/smart-pucrs/DOMAP
[2] http://jacamo.sourceforge.net/

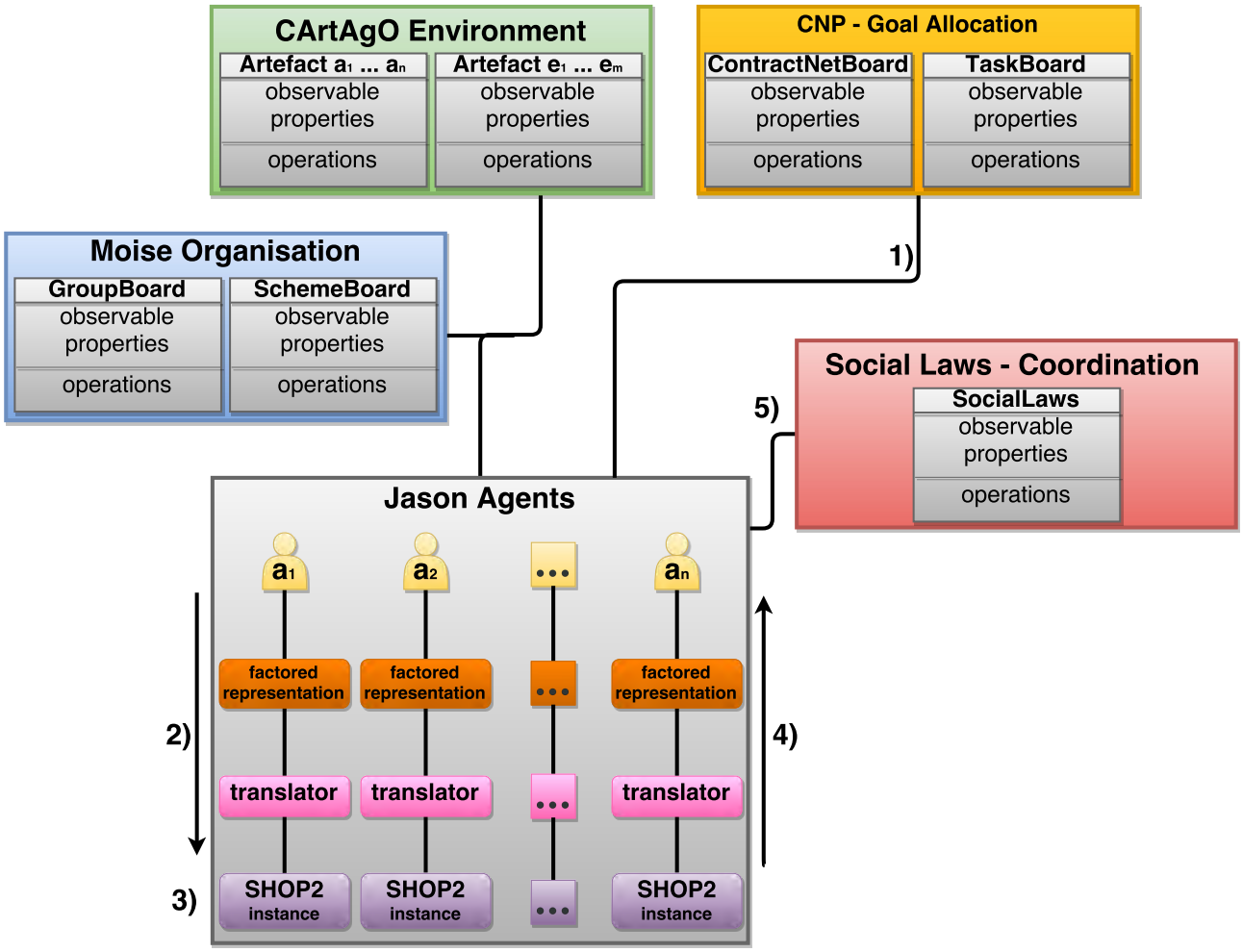

Figure 5.1 – DOMAP runtime overview.

lic environment artifact. `Artifact` e_1 `...` e_m represent environment artifacts. The `GroupBoard` and `SchemeBoard` artifacts have already been discussed in Section 2.2.3.

As mentioned in Section 4.1.1, we assume that agents will not try to access information that was not expressively assigned to them. This is relevant to our implementation because of some of the limitations in the latest version of CArtAgO that we used. CArtAgO artifacts are public, thus, if an agent tries to focus on an artifact, it will succeed and be able to access it. Furthermore, an artifact cannot generate different perceptions for different agents.

There are three internal actions available to start DOMAP's planning — internal actions are actions that Jason agents can execute internally, as opposed to external actions that are related to the environment. The following internal actions are available:

- `DOMAP.plan(SG,TO,OG)`: where `SG` are social goals, `TO` is the time-limit (timeout) for planning, and `OG` is the organisation group that will participate. DOMAP only starts when all agents that belong to this group execute this action.
- `DOMAP.replan(SP,SG,TO,OG)`: drops all social goals from the social plan `SP` and their related intentions. All remaining social goals that have not yet been achieved are announced as new social goals.
- `DOMAP.privateplan(PG,TO)`: This action initiates only the individual planner of DOMAP for this agent in order to solve its private goals `OG`, while still respecting the time-limit `TO`. This is effectively single-agent planning.

In the following sections we discuss the specifics on each of the new elements introduced in Figure 5.1, categorising them into their respective mechanisms.

5.2 Multi-Agent Factored Representation

Agents' use their factored representation interface in order to extract information from plans in its plan library and from CArtAgO environment artifacts. The correlations between JaCaMo elements, our factored representation, and HTN entities are expressed in Table 5.1, an extension of Table 4.1.

We use an HTN translator to parse information from the factored representation interfaces to SHOP2. The translator generates a problem and domain representations for each agent based on information available in their factored representation interface. Environment information is collected from CArtAgO artifacts observable properties into facts and initial states in SHOP2 syntax. Social goals are retrieved from their respective announcements by the organisation, which may come from failed social schemes in Moise. Operators are created from all of the artifact operations that the

Table 5.1 – Correlations between different representations.

JaCaMo elements	**factored representation**	**HTN formalism**
observable property	belief	state
plan	plan	non-primitive task
plan's preconditions	plan's preconditions	non-primitive task constraints
operation	action	primitive task
operation's preconditions	action's preconditions	primitive task's precondition list
operation's effects	action's effects	primitive task's add and delete list
missions	social goals	initial task network

agent has access to; their preconditions are obtained from any conditional tests in an artifact operation; and their delete and add lists are acquired from deletion and addition of observable properties, respectively. Methods are generated from all relevant plans found in that agent's plan library, with the preconditions parsed from the context of the plan, and the task list parsed from actions and subgoals found in the body of the plan.

5.3 Contract Net Protocol

Contract net protocol artifacts [28] mediate the goal allocation phase of DOMAP, with two artifacts: *TaskBoard* and *ContractNetBoard*. Agents that will participate in the goal allocation take the roles of *bidders*. The *bidders* should always focus (agents that focus on an artifact automatically receive any new beliefs/observable properties from that artifact) on the *TaskBoard*, as that is the artifact where social goals are announced. When the *initiator* announces a contract for a new social goal, it creates a *ContractNetBoard* associated with that goal.

We show the observable properties and operations of the *TaskBoard* artifact in Figure 5.2a. When a new new task is announced by the *initiator*, a *task* observable property is created, which is then added as a belief to any agents that are focusing on the artifact. Consequently, a Jason belief addition event is generated that activates the associated plan to focus on the new artifact for the auction of that goal, by using the name value obtained from the observable property, that is, the value is the name of the *ContractNetBoard* created. The *TaskBoard* artifact is linked with the *GroupBoard* artifact, allowing the organisation to perform linked operations described in the link interface.

A link interface includes the set of operations that can be executed by other artifacts. Thus, link operations cannot be accessed by agents, but only by linking artifacts. Therefore, only the *GroupBoard* artifact, operated by the *initiator*, can announce goals in the *TaskBoard*. Once the auction process is finished, the *initiator* performs the *clear* operation to delete the observable property associated with that goal. This gener-

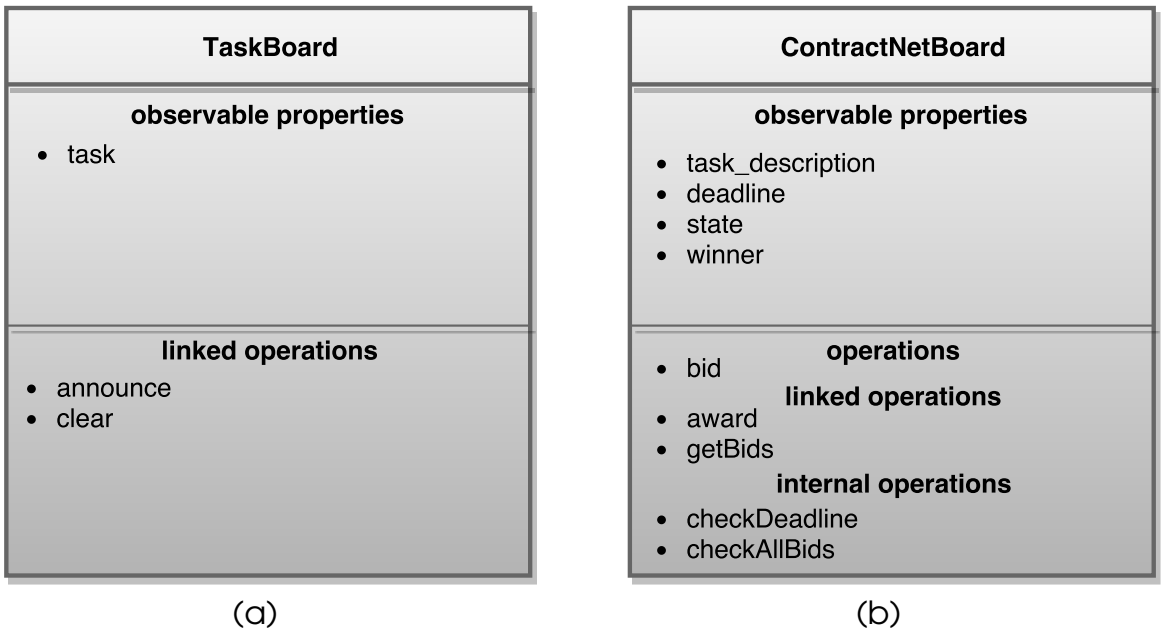

Figure 5.2 – (a) The task board artifact; (b) The CNP board artifact.

ates a belief deletion event in Jason, to allow agents to perform clean-up and any other necessary activities.

The observable properties and operations of the *ContractNetBoard* artifact can be observed in Figure 5.2b. A *ContractNetBoard* is created for each goal announced by the *initiator*. The observable properties of *ContractNetBoard* are as follows:

- *task_description* contains a social goal that is the auction's prize.
- *deadline* is the time, in milliseconds, that the auction will run for.
- *state* informs if the auction is open or closed, it starts with the open value.
- *winner* is created by the *initiator* once the auction ends, containing the ID of the bid that won the auction.

The operations of *ContractNetBoard* are as follows:

- *bid* is executed by *bidders* in order to place a bid for the goal associated with the artifact.
- *award* is a linked operation executed by the *initiator*, it updates the *winner* observable property, based on a value function (described in Section 4.1.2).
- *getBids* is another linked operation executed by the *initiator*, it returns all bids currently placed.

The creation of the *winner* observable property results in a Jason belief addition event to all of the agents that are currently focusing on the artifact. The parameter of

winner is the bid ID of the agent who won the auction. The bid ID is shown instead of the winner's name in order to preserve privacy, since agents only know their own bids' ids. Thus, an agent needs to compare the ID from the *winner* parameter to his bid ID, for that particular artifact, in order to determine if it won.

There are also two internal operations, *checkDeadline* and *checkAllBids*. The *checkDeadline* operation closes bidding if the deadline is past. The *checkAllBids* operation closes bidding if all agents have already placed their bid. Both internal operations update the *state* observable property to closed.

5.4 SHOP2

We have shown the basics of HTN planning and SHOP2 in Section 2.1.2 and Section 4.1.3. The SHOP2 planner is written in Common Lisp and requires a Common Lisp implementation in order to work. We use the Allegro CL Free Express Edition[3] to run SHOP2 in DOMAP. Although other implementations are also supported by SHOP2 (such as Steel Bank CL, Clozure CL, and GNU clisp) we only provide Java functions and Allegro scripts to run SHOP2 in DOMAP with JaCaMo. These scripts are listed in Appendix B.

5.5 Social Laws

We discussed our model of social laws in Section 4.1.4, now we discuss how we implemented it as a CArtAgO artifact. This artifact is responsible for coordinating agents at runtime in order to avoid any conflicts.

In Figure 5.3, we show the observable properties and operations of the *SocialLaws* artifact. This artifact is created during the system's initialisation, one instance for each of the social laws. Agents consult the *SocialLaw* artifact associated with the action that they are about to execute. This process is only necessary for actions that are part of social plans, and only if those actions are annotated with conflict flags in the factored representation.

Its observable properties are the following:

- *social_law* contains the name of the social law.
- *action_name* is the name of the action that is associated with this social law.
- *precondition_list* is the list of preconditions that make this social law applicable.

[3] https://franz.com/downloads/clp/download

- *action_options* contains the behaviour that an agent has to follow in order to avoid any conflicts.

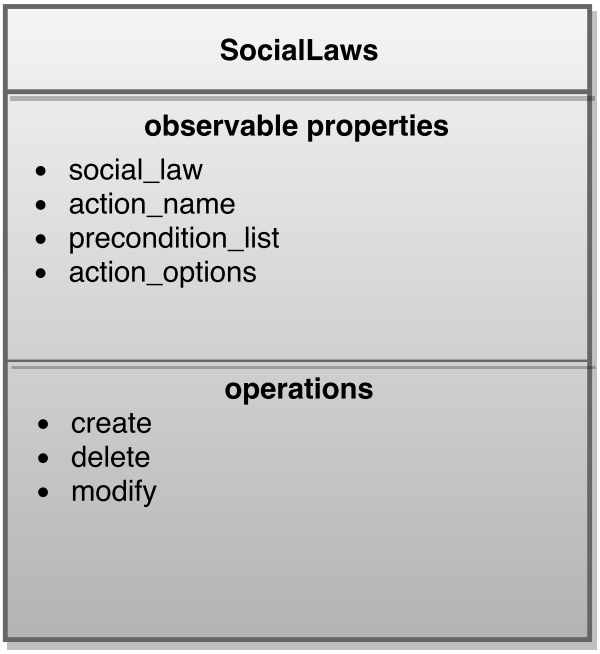

Figure 5.3 – The artifact for social laws.

We also provide operations related to the synthesis of social laws; although there is no mechanism implemented to make use of these operations, it allows external mechanisms for synthesis of social laws to be used. The *create* operation allows the creation of another instance of *SocialLaws*. The *delete* operation erases the current instance of *SocialLaws*. And the *modify* operation permits to alter the values of the observable properties in the instance of *SocialLaws* that the operation was invoked.

In this chapter, we described how we implemented the DOMAP framework using the JaCaMo MAS development platform. In the next chapter, we show the evaluation of DOMAP across three different domains and against four state-of-the-art multi-agent planners.

6. DOMAP'S EVALUATION

To the best of our knowledge, DOMAP is the only recently developed multi-agent online planner that is able to take advantage of the different programming abstractions in MAS (agent, environment, and organisation). Thus, in order to evaluate our framework, we isolated the online components of DOMAP, and ran offline experiments comparing against four state-of-the-art multi-agent offline planners (described in Section 2.3.2), all of which took part in the 2015 Competition of Distributed and Multi-Agent Planners (CoDMAP-15) [138]. *SIW+ -then-BFS(f)* [85] was the top performing planner with regards to planning time, out of 17 planners. *CMAP-t* [16] obtained second place, *ADP-legacy* [37, 35] third place, and *PMR* [77] eighth place.

We summarise differences between some features of these planners in Table 6.1. Planning in PMR depends on the planner selected, it is decentralised if the first planner succeeds in finding a plan, otherwise (i.e., if replanning is required) it uses a centralised single-agent planner. DOMAP is decentralised even when replanning is necessary. DOMAP and ADP-legacy are the only ones that can result in different goal allocations for the same problem, and as such, can offer different solutions depending on these allocations. Although DOMAP is the only one to use HTN rather than PDDL, which can benefit from extra domain information, we limited it to contain only information that was also available in the PDDL descriptions of the other planners. Both DOMAP and SIW+ -then-BFS(f) decouple domain and problem information from these formalisms and use "agnostic" data structures to represent them as well (DOMAP uses the factored representation).

Table 6.1 – Features of multi-agent planners used in the experiments.

	planning	**multi-core**	**formalism**	**goal allocation**	**replan**
ADP-legacy	centralised	no	PDDL	yes (non-deterministic)	no
CMAP-t	centralised	yes	PDDL	yes	no
DOMAP	decentralised	yes	HTN	yes (non-deterministic)	yes
PMR	both	yes	PDDL	yes	yes
SIW	centralised	no	PDDL	no	no

All of the multi-agent domains used in CodMAP-15 were simplistic adaptations of single-agent planning domains from past editions of the IPCs, thus not the kind of domains that are usually found in MAS. We chose one of their adaptations, which was the classical Rovers domain, and ran some experiments. Our results were very similar to all other planners from the competition; since their problems were all relatively small scale, planning would often finish very quickly. When the difference between results are a few milliseconds, it is hard to claim that one planner is really faster than the others.

Thus, we chose 10 of the 20 problems in the Rovers domain from the competition, and scaled them up appropriately, increasing the number of agents and goals. The

lack of complex multi-agent domains also led us to design a new domain, we call it the Floods domain (Chapter 3), with more characteristics of MAP and MAS, such as heterogeneous agents and multiple types of goals. The third domain used in our experiments was selected from ICKEPS[1] 2012, the Petrobras domain [135].

Each domain had 10 problem variations, increasing the number of agents, goals, and initial state literals. We gave all planners a time limit of 60 minutes for each problem. We ran each of the five planners 20 times for the 10 problems in each domain, resulting in a total of 2000 executions, 400 for each planner. For time measurements, we extracted the average, minimum, and maximum time spent planning (goal allocation times included). We used real time rather than CPU time, since some planners required extra translations and data preparation before planning, and also because some planners used multiple cores, so CPU time would not be appropriate unless we used only the highest time from one of the cores.

We also extracted the average, minimum, and maximum plan cost for all solutions found, as a possible measurement of plan quality. Plan cost is equal to plan size, since all actions are assumed to have unit cost. Our third measurement is what we call parallelisation, which we also use to measure plan quality. Given that agents will concurrently execute the plans, this is a very important measure to be considered in multi-agent execution. Parallelisation is defined by the variance of the plan cost of each individual agent, thus indicating how much the actions are spread across all agents (i.e., if the loads are balanced). Depending on the domain, either plan size or parallelisation will be the most indicative measure of plan quality. Plan size works best for measuring plan quality in domains with single, or a low number of, agents and domains that require sequential actions. Parallelisation, as the name suggests, is best in domains with many parallel or concurrent actions. For the three domains used in this thesis we will show that parallelisation is the most important metric for plan quality, however, we include results for plan size as a comparative.

Our final measurements are the average, minimum, and maximum time spent executing the generated plans. Because the other four planners do not offer any means for executing their solutions, we translated them into JaCaMo and ran the solutions found by each planner as a MAS. Only the floods domain contains conflicts, and to keep this process fair we removed it from the domain description of other planners, and used DOMAP coordination mechanism to solve conflicts at runtime for all planners. Since we are not using any temporal planner, we set the same execution time of 500 milliseconds to all actions.

The environment in the planning phase is assumed to be deterministic, however, in the execution phase the environment can be non-deterministic and actions can fail, in which case it may be necessary to replan accordingly. Each agent has its own

[1] http://icaps12.icaps-conference.org/ickeps.html

perspective of the environment, and thus a single agent might not have complete information about the environment. The computer we used to run the experiments[2] has the following specification: Intel Xeon Processor E5645 (12M Cache, 2.40 GHz, 6 cores, 12 threads), 32 GB of memory, Ubuntu 16.04 operating system, and Java 8. In the following sections we describe the settings, results, and discussion regarding our experiments in these three domains.

6.1 Rovers Domain Experiments

Rovers is a classical domain in automated planning, dating back to 2002 when it first appeared in the 3rd International Planning Competition (IPC). It is inspired by rover vehicles, commonly used by NASA in space missions on Mars during that period. In the Rovers domain, these rover vehicles are equipped with different capabilities to explore Mars' surface collecting samples and taking pictures of objectives, and then transmitting all data back to a lander spaceship. This domain does not have any conflicting actions or dependencies. Problems in the Rovers domain can have three different types of goals: communicate soil, rock, or image data from a specific waypoint.

There are nine different actions available to rover vehicles. A navigate action that can move a rover from waypoint W to W', as long as the vehicle is able to traverse from W to W'. An action to sample soil and an action to sample rock, requiring that the vehicle be equipped with the appropriate tools and have an empty store (a store is full if it has any type of sample on it). There is an action to drop any sample that is in the store. An action to calibrate a camera using an objective as focus, which requires the vehicle to be equipped for imaging, have a camera, be located in a waypoint from where the objective is visible from, and for the objective to be a calibration target for the camera. The action to take an image of an objective using a specific camera mode (low resolution, high resolution, or coloured) needs for the vehicle to be equipped for imaging, have a calibrated camera that supports the appropriate mode, and for the vehicle to be located in a waypoint from where the objective is visible from. The last three are communication actions that can send rock, soil, or image data back to a lander spaceship, if the vehicle has the data and is in range of the lander.

6.1.1 Setting

Rovers' multi-agent adaptation from CodMAP-15 was restricted to defining which predicates were private and could not be shared between agents, and establishing

[2]The source code for these experiments can be found at https://github.com/smart-pucrs/DOMAP

that each rover vehicle is an agent. From the 20 problems used in the competition, the largest had 10 agents and 20 goals. For a loosely-coupled domain, this is too low, since multi-agent planners are especially fast in these domains, where coordination can be kept at a minimum. We chose 10 of their 20 problems, and increased considerably the number of agents and goals per problem. Problem 1 starts with 4 agents and 8 goals, and each problem after it adds 2 agents and 4 goals, hence Problem 10 has 22 agents and 44 goals, more than double the size of the biggest problem used in the competition. The configuration of each problem is shown in Table 6.2.

Table 6.2 – Rovers problem configurations.

	Rovers	**Waypoints**	**Cameras**	**Soil data goals**	**Rock data goals**	**Image data goals**
p01	4	7	6	3	3	2
p02	6	8	4	4	4	4
p03	8	10	5	6	5	5
p04	10	12	4	7	6	7
p05	12	20	7	5	9	10
p06	14	25	7	10	10	8
p07	16	40	11	12	10	10
p08	18	25	10	12	14	10
p09	20	30	7	18	12	10
p10	22	45	15	18	15	11

6.1.2 Results

In Figure 6.1, we show the results for minimum, maximum, and average planning time in the Rovers domain. Time in seconds is on the y-axis and in logarithmic scale to improve readability. The first five problems have a slow increase in planning time across all planners, but they all have similar performances, with differences in the order of milliseconds. One of the 20 runs from DOMAP for the first, fourth, fifth, and sixth problem suffered from poor goal allocation, which increased the average, but most other runs were much faster, which can be observed in the minimum results graph.

SIW+ -then-BFS(f) could not solve problem 8 under the time limit of 60 minutes. ADP-legacy had some of the best times for the first six problems, but it is followed closely by most other planners. SIW+ -then-BFS(f) scales poorly when increasing the number of agents and goals in each problem. In the last four problems, ADP-legacy and PMR performances drop considerably, while CMAP-t stays competitive with DOMAP, but does not seem to scale quite as well.

The size of plans found in solutions from each planner can be observed in Figure 6.2. DOMAP has good plan sizes in problems 1–5, but in problems 6, 9, and 10,

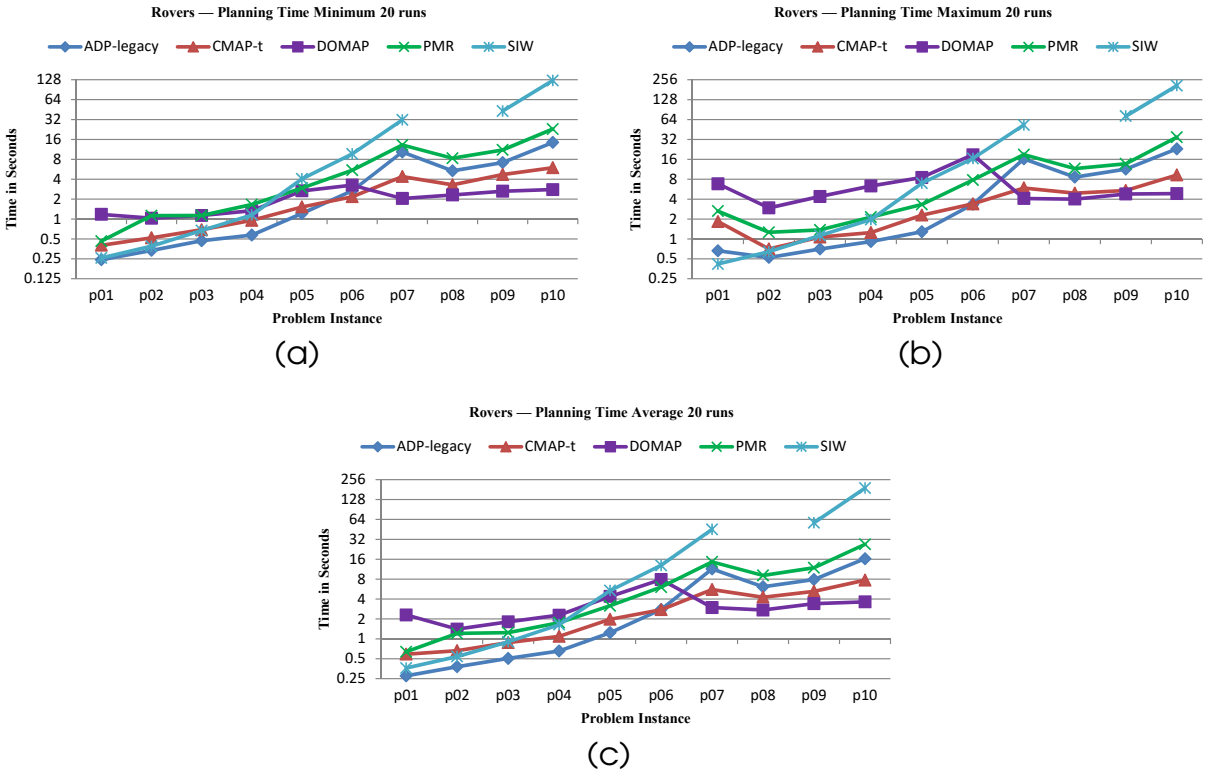

Figure 6.1 – (a) Minimum time spent planning; (b) Maximum time spent planning; (c) Average time spent planning.

DOMAP's plan length scales very poorly, while the other planners remain competitive across all problems.

In Figure 6.3, DOMAP achieves the best performance in regards to parallelism, especially when considering the minimum values found in 20 runs. SIW+ -then-BFS(f) has good parallelisation in problem 6, with the minimum plan variance of 72 against DOMAP's second best 91.

The time spent in execution is shown in Figure 6.4. DOMAP once again shows excellent scalability and the best performance overall, while other planners are inconsistent and their execution times fluctuate depending on the problem.

Finally, in Figure 6.5, time spent planning is combined with time spent executing in bar graphs. SIW+ -then-BFS(f) is not included in Figure 6.5b because its total time was much higher than other planners and would affect readability. In the first five problems, performance is comparable across all planners, but in problems 8, 9, and 10 (i.e., the largest problems), DOMAP has the best results.

Appendix C contains tables with all values used to generate each of the graphs found in this section.

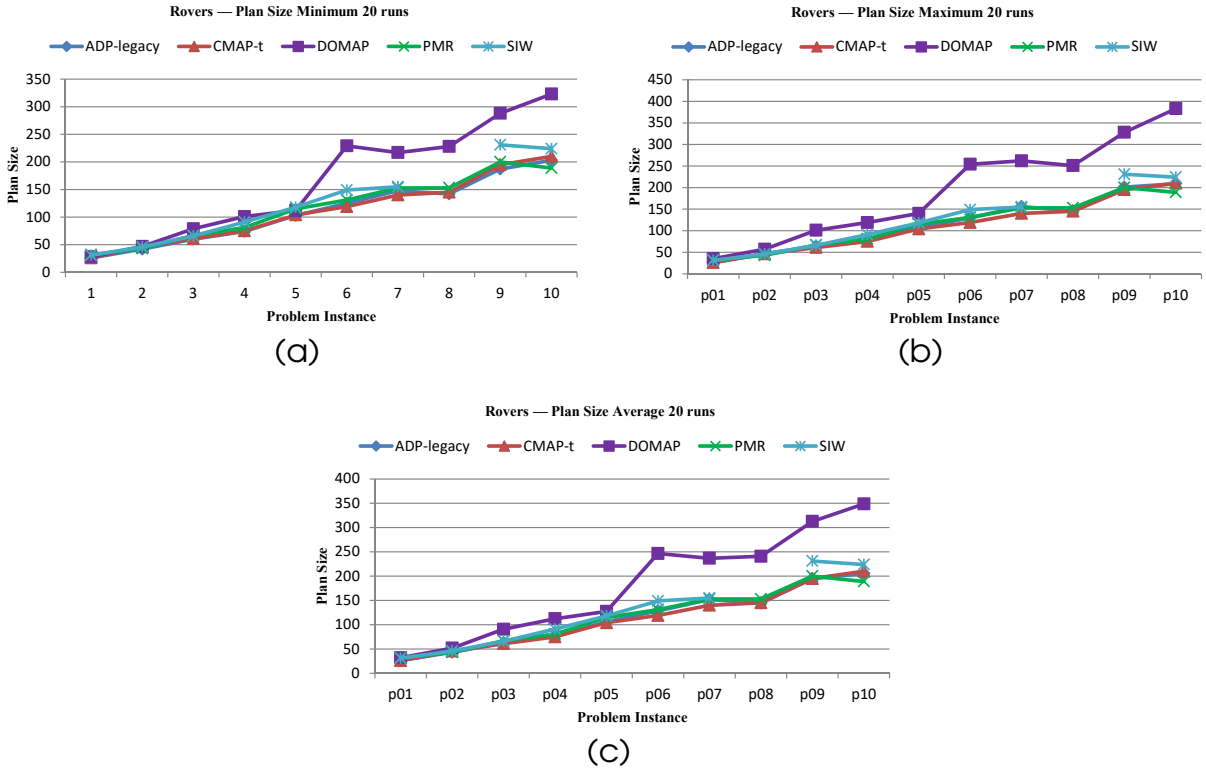

Figure 6.2 – (a) Minimum plan size; (b) Maximum plan size; (c) Average plan size.

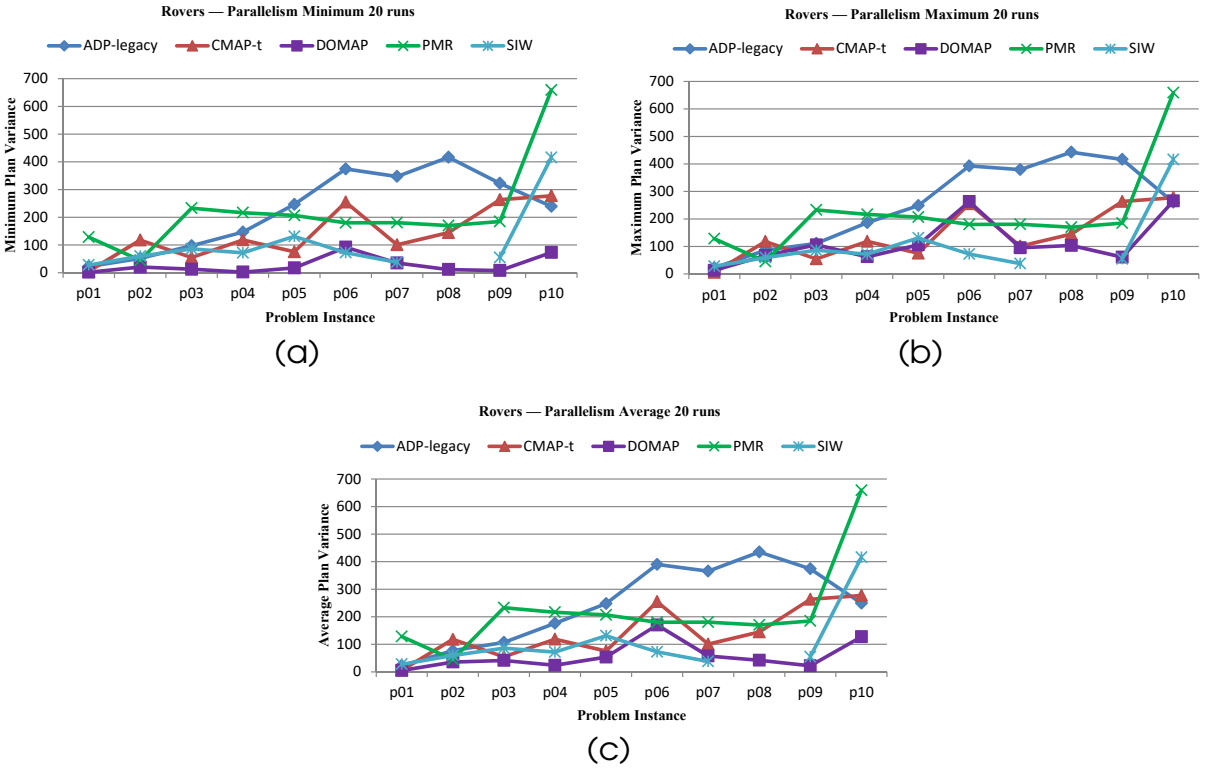

Figure 6.3 – (a) Minimum plan size variance between all agents; (b) Maximum plan size variance between all agents; (c) Average plan size variance between all agents.

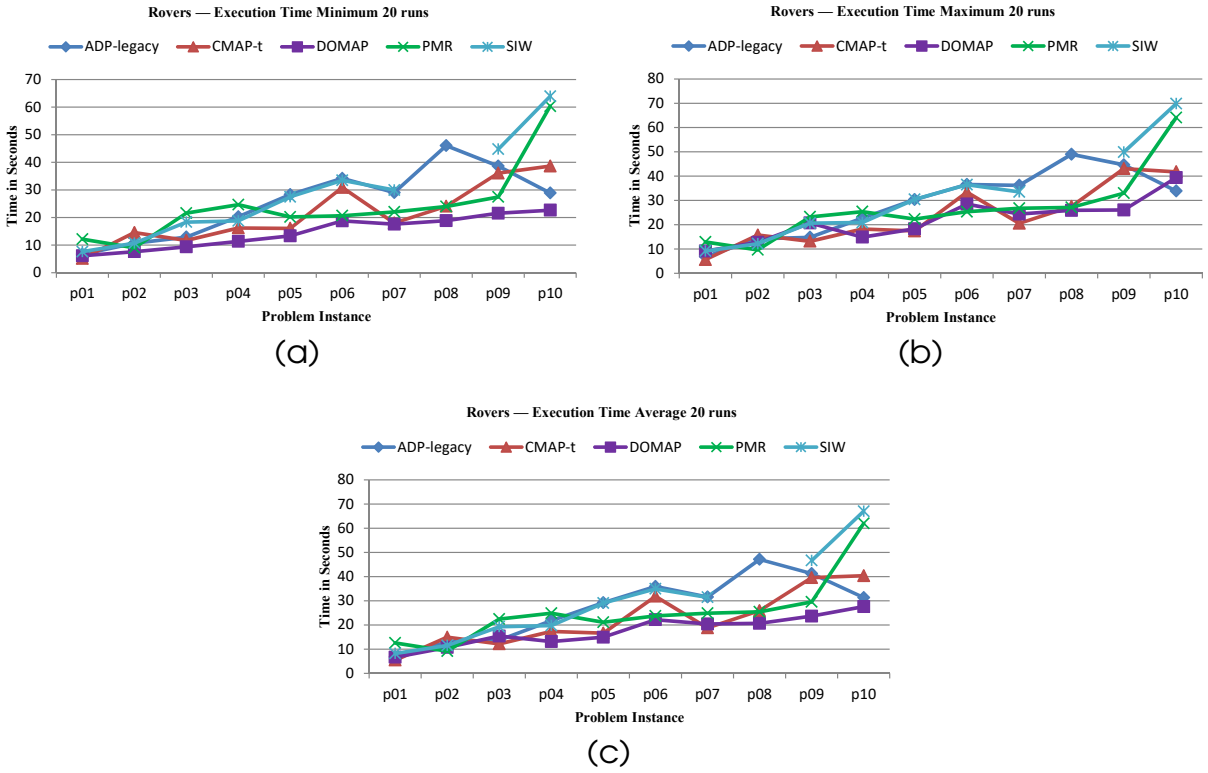

Figure 6.4 – (a) Minimum time spent executing solutions; (b) Maximum time spent executing solutions; (c) Average time spent executing solutions.

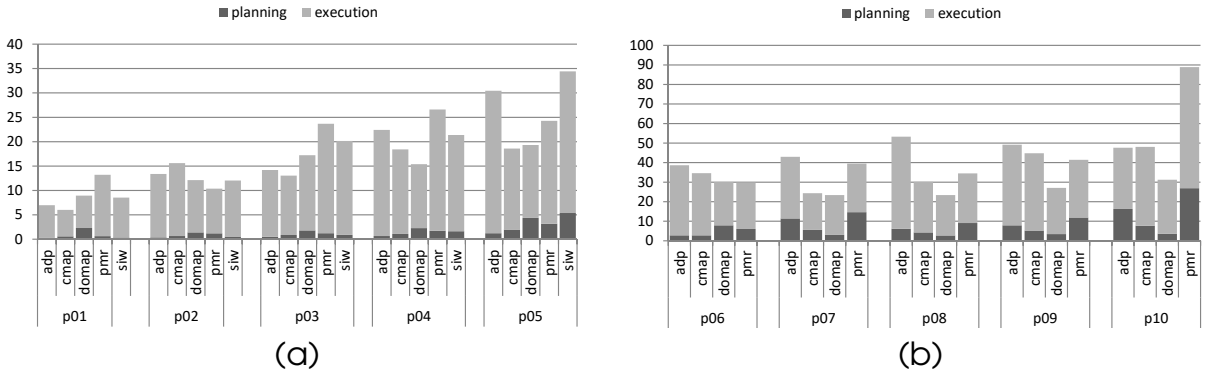

Figure 6.5 – (a) Planning and execution times for the first 5 problems in the Rovers domain; (b) Planning and execution times for the last 5 problems in the Rovers domain.

6.1.3 Discussion

The Rovers domain is not a particularly good domain for DOMAP because agents are very homogeneous, their capabilities often overlap, and many goals can only be solvable by a handful of specific agents, due to each rover having their own traversable paths. Problems 5 and 6 had the highest maximum planning time for DOMAP, since all 20 runs in both problems required replanning; problem 5 varied between 2 and 3 rounds of replanning, and problem 6 between 3 and 4 rounds. Regardless of this disadvantage, DOMAP outperforms other planners in the last four (largest) problems, even though there were some runs where DOMAP had to replan a few times.

DOMAP's poor performance in some of the first problems are mainly because it requires some initial start-up time, since it is started as a MAS. Another factor is the use of CNP; while it allows for decentralised goal allocation, it also requires a certain amount of time for agents to be able to calculate and place their bids, and then select the winners. We used a deadline of 50 milliseconds for agents to calculate their bids, but because we are running agents in the same computer, the actual time spent calculating bids may vary depending on thread scheduling.

DOMAP's high plan size is a consequence of the fairness heuristic used in goal allocation, and since it is non-deterministic, it is possible to have large variations in each different run. By prioritising fairness in goal allocation (allocating goals to agents as evenly as possible), plan size tends to increase as more agents are used and they have to do more actions than if using less agents. However, as our execution results have demonstrated, plan size is not the best plan quality metric for multi-agent planning. Although many multi-agent planners, including CMAP and PMR, present makespan (the largest plan size found in any individual agent) as an alternative metric for plan quality, it is not representative of execution performance, where the most important factor is to have as many concurrent actions as possible. Our parallelisation metric captures this factor, and as shown in our results have a direct impact in execution performance, where DOMAP excels.

As previously stated, ADP-legacy and DOMAP use non-deterministic goal allocation mechanisms that can vary significantly across runs. Thus, 20 runs might not have contained the best- and worse-case allocations.

6.2 Floods Domain Experiments

The Floods domain was introduced in Chapter 3, in this section we provide a brief summary. Our domain was inspired by a real-world scenario on using a multi-robot team for search and rescue operations after flooding disasters. Differently from

the Rovers domain, we designed Floods to have pre-established roles, which are usually found in MAS organisations. In the Floods domain, a team of heterogeneous autonomous robots are dispatched to monitor flooding activity and execute search and rescue operations within a geographical *region*, which is divided into several interconnected *areas*. The Centre for Disaster Management (CDM) establishes bases of operation in the region that is being monitored. These bases are used to receive and interpret data about floods and victims found by the robots. The CDM is usually operated by humans, but in our JaCaMo+DOMAP implementation we simulate them by using agents, capable of creating dynamic goals at runtime.

There are three different roles of autonomous vehicles: *Unmanned Aerial Vehicle (UAV)*: aerial units that have no movement restrictions, but are only equipped to capture images of flooded areas; *Unmanned Ground Vehicle (UGV)*: ground units that can only move through areas connected by *ground paths*, they can capture images, and are also capable of delivering first aid kits; *Unmanned Surface Vehicle (USV)*: naval units that can move through areas connected by *water paths*, they can capture images, and also collect water samples. During flood events, these vehicles are required to capture images and transmit them back to a *CDM*. Analysis of these images can identify potential victims that are stranded or in danger, and deploy first responders who may request additional *first aid kits* to be sent to their location to help these victims. *Water samples* may also be requested to be collected from certain flooded areas.

When more than one USV tries to move from an area A to an area A', it causes a conflict. This conflict represents the narrowness of water paths, where only one USV may pass through a path at a time. We use our domain-independent *priority law* to solve this conflict. When the social law artifact detects the conflict (two or more USV agents sent their move action from A to A'), an agent is chosen arbitrarily to proceed with its action, while the others adopt the behaviour specified in the social law, to hold their action.

6.2.1 Setting

Because we created Floods problems from scratch, we had a lot more control over how each problem scaled. Problem 1 starts with 9 agents (3 UAV, 3 UGV, and 3 USV), 15 areas, 2 CDMs, and 9 goals (a combination of taking pictures of flooded areas, delivery of first aid kits, and collection of water samples). Each subsequent problem added 3 agents (1 of each role), 5 areas, and 3 goals. Every other problem had one CDM added. The last problem, p10, has 36 agents, 60 areas, 6 CDMs, and 36 goals. The configuration of each problem is shown in Table 6.3.

Table 6.3 – Floods problem configurations.

	UAVs	UGVs	USVs	Areas	CDMs	Flood disasters	Boxes	Water samples
p01	3	3	3	15	2	5	2	2
p02	4	4	4	20	2	6	3	3
p03	5	5	5	25	3	7	4	4
p04	6	6	6	30	3	8	5	5
p05	7	7	7	35	4	9	6	6
p06	8	8	8	40	4	10	7	7
p07	9	9	9	45	5	11	8	8
p08	10	10	10	50	5	12	9	9
p09	11	11	11	55	6	13	10	10
p10	12	12	12	60	6	16	10	10

6.2.2 Results

In Figure 6.6, we show the results for planning time across 20 runs of each problem. Time in seconds is on the y-axis and in logarithmic scale to improve readability. SIW+ -then-BFS(f) had very large planning times, up to an average of 3135 seconds for problem 10. CMAP-t has lower planning times for most of the problems, except for the largest problems, where DOMAP performs best.

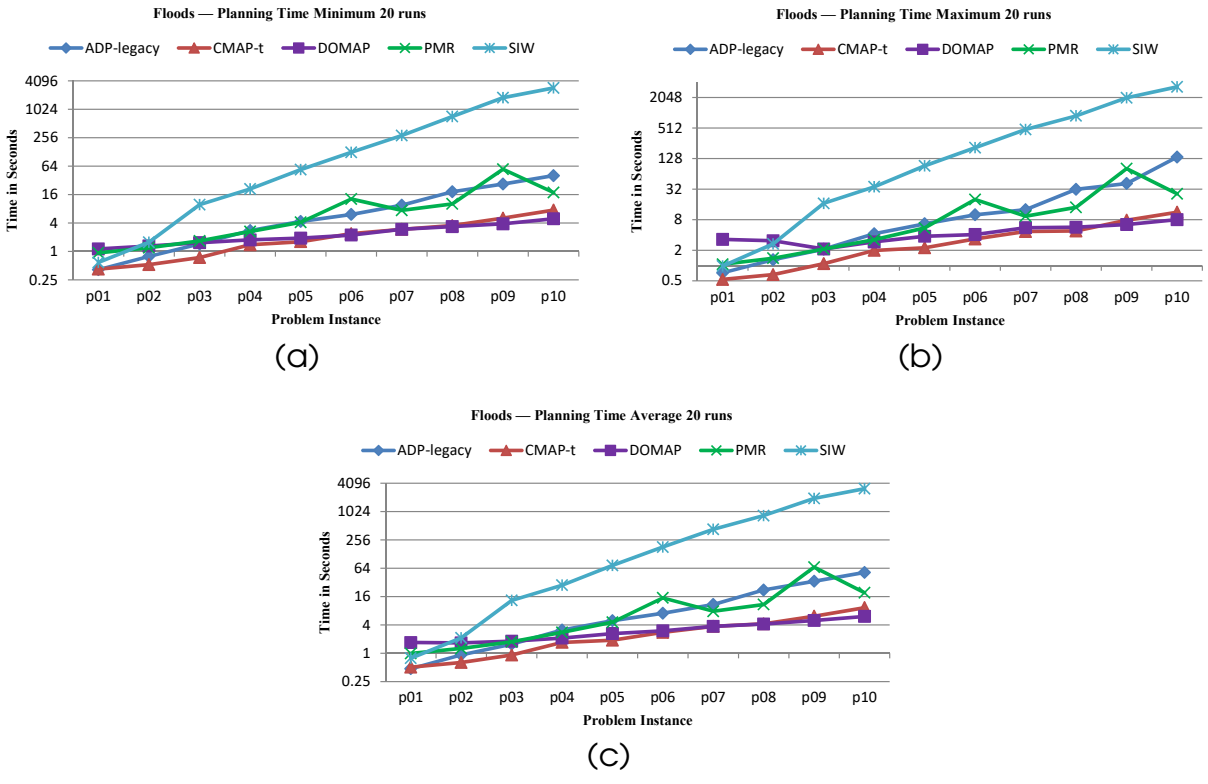

Figure 6.6 – (a) Minimum time spent planning; (b) Maximum time spent planning; (c) Average time spent planning.

Plan size is shown in Figure 6.7. CMAP-t, ADP-legacy, and PMR found some of the lowest cost plans. PMR does much worse on problems 6 and 9, when it has to use multiple planners, but it is able to find the lowest cost plan for the largest problem (p10). DOMAP and SIW+ -then-BFS(f) have mediocre average plan lengths when compared to other planners for the largest problems. Results of plan length were much more spread out than the planning time results, with each planner generating the lowest cost plan for at least one problem. When considering minimum plan length found, DOMAP manages to improve its plan size considerably.

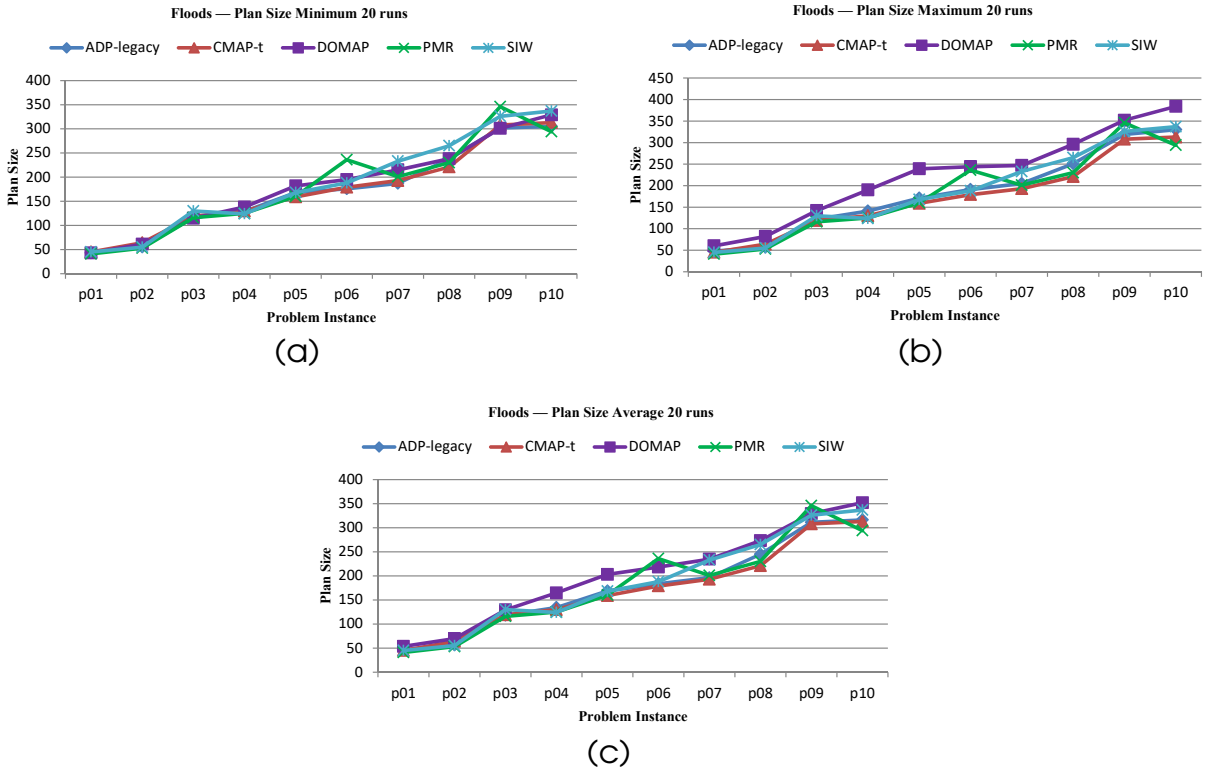

Figure 6.7 – (a) Minimum plan size; (b) Maximum plan size; (c) Average plan size.

DOMAP's goal allocation, of prioritising fairness among agents pays off when we consider concurrent actions, as can be seen in Figure 6.8. Our results show that DOMAP has excellent parallel solutions to all problems, dominating other planners on most of the problems.

Time spent in execution is shown in Figure 6.9. DOMAP is the only one that scales reliably, possibly due to its good use of fairness during goal allocation. Even though DOMAP solutions had, on average, some of the longest plans, because it distributed social goals as evenly as possible, it still managed to finish execution faster than all other planners in most of the problems.

Figure 6.10 combines the time spent planning and executing for all ten problems in the Floods domain. We removed SIW+ -then-BFS(f) from these graphs for readability, as its planning times for these problems were too high. The results for the first

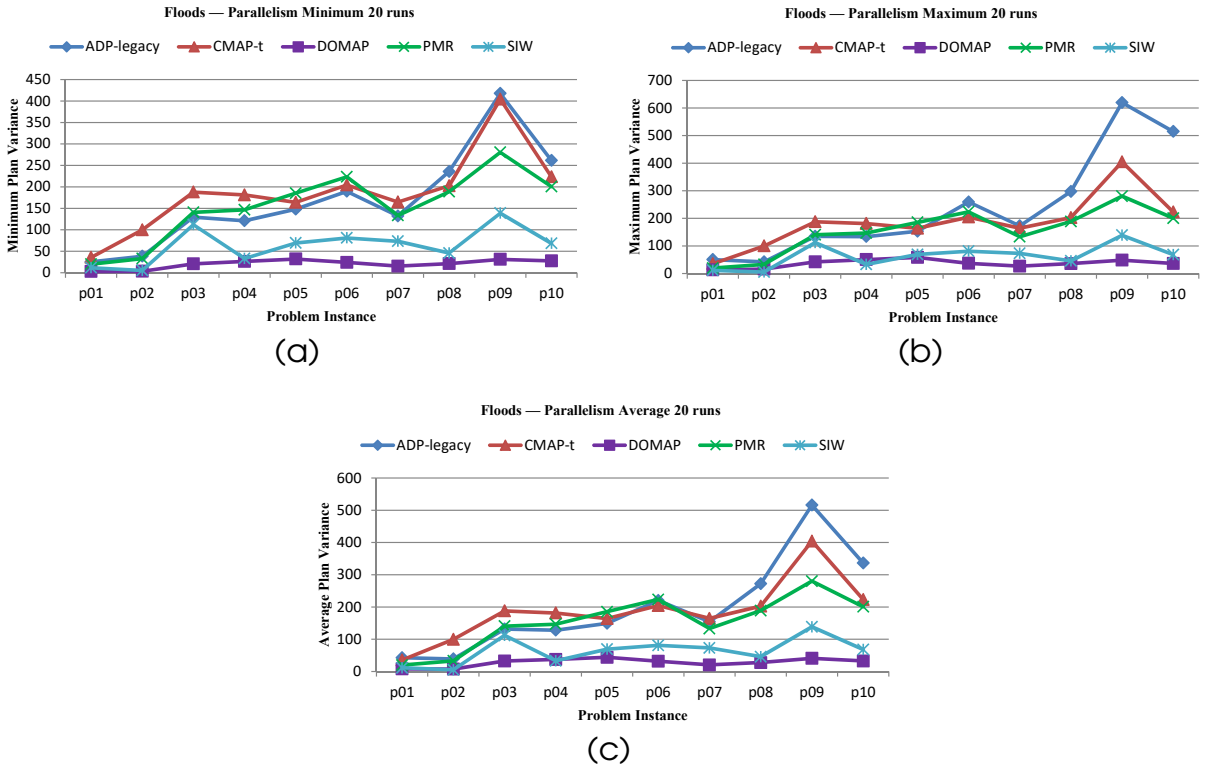

Figure 6.8 – (a) Minimum plan size variance between all agents; (b) Maximum plan size variance between all agents; (c) Average plan size variance between all agents.

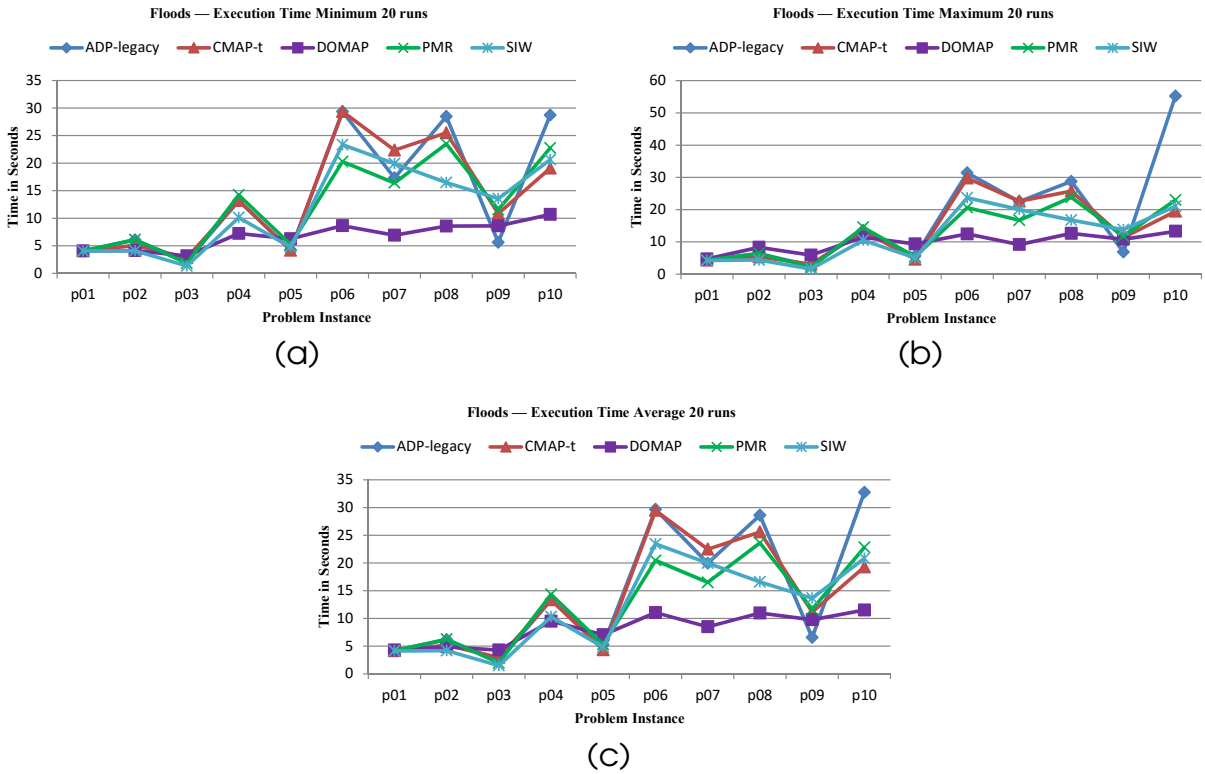

Figure 6.9 – (a) Minimum time spent executing solutions; (b) Maximum time spent executing solutions; (c) Average time spent executing solutions.

five problems (Figure 6.10a) were very similar across the other four planners, since they had comparable planning and execution times. When combining planning and execution, DOMAP appears to scale well, obtaining the best results. CMAP-t, for the most part, is able to follow closely in some problems, while the other planners do not perform as well.

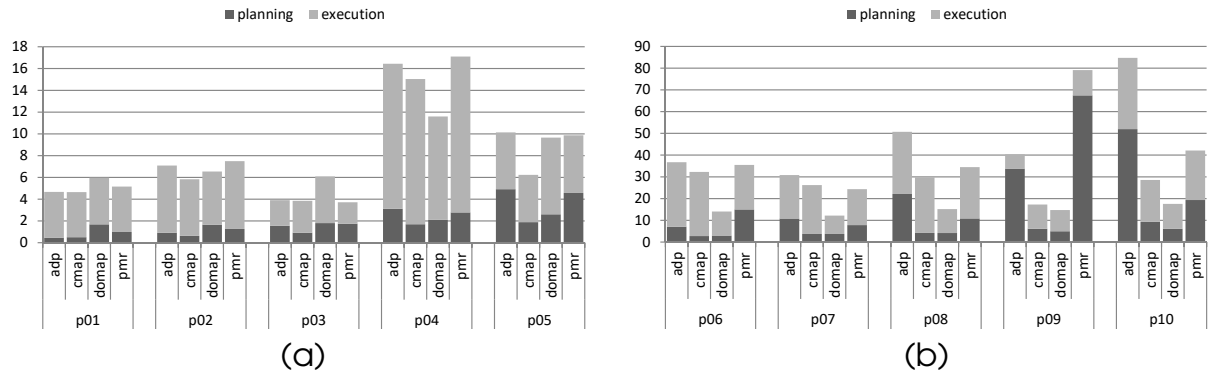

Figure 6.10 – (a) Planning and execution times for the first 5 problems in the Floods domain; (b) Planning and execution times for the last 5 problems in the Floods domain.

Appendix D contains tables with all values used to generate each of the graphs found in this section.

6.2.3 Discussion

SIW+ -then-BFS(f), the winner of CoDMAP-15 with regards to planning time, performed the worst in our planning time experiments. This makes sense since it is the only planner that does not separate goal allocation and planning, thus being unable to cope with the increasing planning complexity each time the number of agents and goals increase. Their results in the Floods domain were even worse than their results in the Rovers domain (which had a smaller number of agents).

PMR and ADP-legacy are very close for some of the first problems with regards to planning time, but PMR seems to scale better since it makes use of multiple cores, while ADP-legacy does not. In problems 6 and 9, PMR had to switch planners and apply plan reuse, which explains its poor performance when compared to other planners. CMAP-t and DOMAP presented the best planning time results; the former is better in the first problems, while the latter has better results for the last problems. DOMAP again appears to scale better than all other planners, at least for experiments in loosely-coupled domains.

We also ran experiments with DOMAP banning roles instead of individual agents. In both cases the maximum amount of replanning rounds were only 1 across all 400 executions (200 for DOMAP with role ban, 200 for DOMAP with agent ban).

Agent bans had at most 1 round because of our bid selection fairness heuristic that gives priority to agents that have not expanded recursive plans, which in this domain are UAVs, who have no movement restrictions and thus are able to more easily find a solution. In other domains, such as in the Rovers domain, this may not be the case, where more than 1 reallocation/replanning rounds were necessary. Because the results were very similar, we omitted DOMAP results with role ban and showed only DOMAP with agent ban, as it is a more domain-independent solution, and it was the same ban strategy that we used in the Rovers domain.

Our results also indicate that allocating goals before planning can lead to a huge improvement with regards to planning time, since SIW+ -then-BFS(f) is the only planner that does not do so. Moreover, decentralising planning (i.e., running an individual planner for each agent, which only DOMAP and PMR do so) and running in computers with multiple cores (only DOMAP, CMAP-t, and PMR take advantage of multiple cores) can also lead to faster planning times. The range between minimum and maximum plan lengths in DOMAP indicates that bid calculation and selection heuristics can be improved to minimise this gap. Finally, by also decentralising goal allocation in DOMAP, we are able to preserve autonomy and privacy of agents, while trying to maximise fairness, at the extra incurred cost in time to allocate goals.

6.3 Petrobras Domain Experiments

The Petrobras domain was first introduced [66] as a challenge domain in the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012), held during ICAPS 2012. It is targeted at modelling planning and scheduling of ship operations on petroleum platforms and ports based on a real problem found in the Brazilian petroleum company Petrobras. The general problem is the transportation and delivery of cargo to a number of different locations, respecting any constraints that are imposed, such as vessel load and fuel capacity. The goal is to optimise execution cost of the resulting schedule.

Since its introduction, there have been a wide variety of work using the Petrobras domain. In [135], it is shown how the domain can be modelled in UML and then translated into PDDL, as well as using the itSIMPLE [134] tool to investigate and describe the efficiency of the model. The limitations of available domain-independent planners with regards to realistic problems in this domain are also shown.

In [128], the authors describe three approaches to solve the Petrobras challenge, using classical planning, temporal planning, and single-player games with Monte-Carlo tree search. Two ideas that were originated from experiences with the Petrobras domain are described in [6]. The use of finite state automaton to describe expected se-

quences of actions with arcs that are annotated by conditions to guide the planner to explore good paths in the automaton, and turning primitive actions into meta-actions to decrease the size of the finite state automaton and improve efficiency of planning. Finally, in [12], a reformulation of the Petrobras domain in SAT is proposed, specifically with satisfiability modulo theories (SMT) encodings, and SMT solvers are used to solve them.

Problems in this domain focus on the transportation of cargo from ports on the land to platforms in the ocean. Two strips of the Brazilian coast are considered, a port in *Rio de Janeiro* and a port in *Santos*, as shown in Figure 6.11 [135]. Each strip has a set of ocean platforms, six in the *Rio de Janeiro* strip ($F1$, ..., $F6$), and four in the *Santos* strip ($G1$, ..., $G4$). There is one waiting area off-shore in each strip ($A1$ and $A2$), where vessels can wait in between deliveries. Ports and platforms $F5$ and $G3$ can refuel vessels. Ports can dock two vessels simultaneously, while platforms can only dock one.

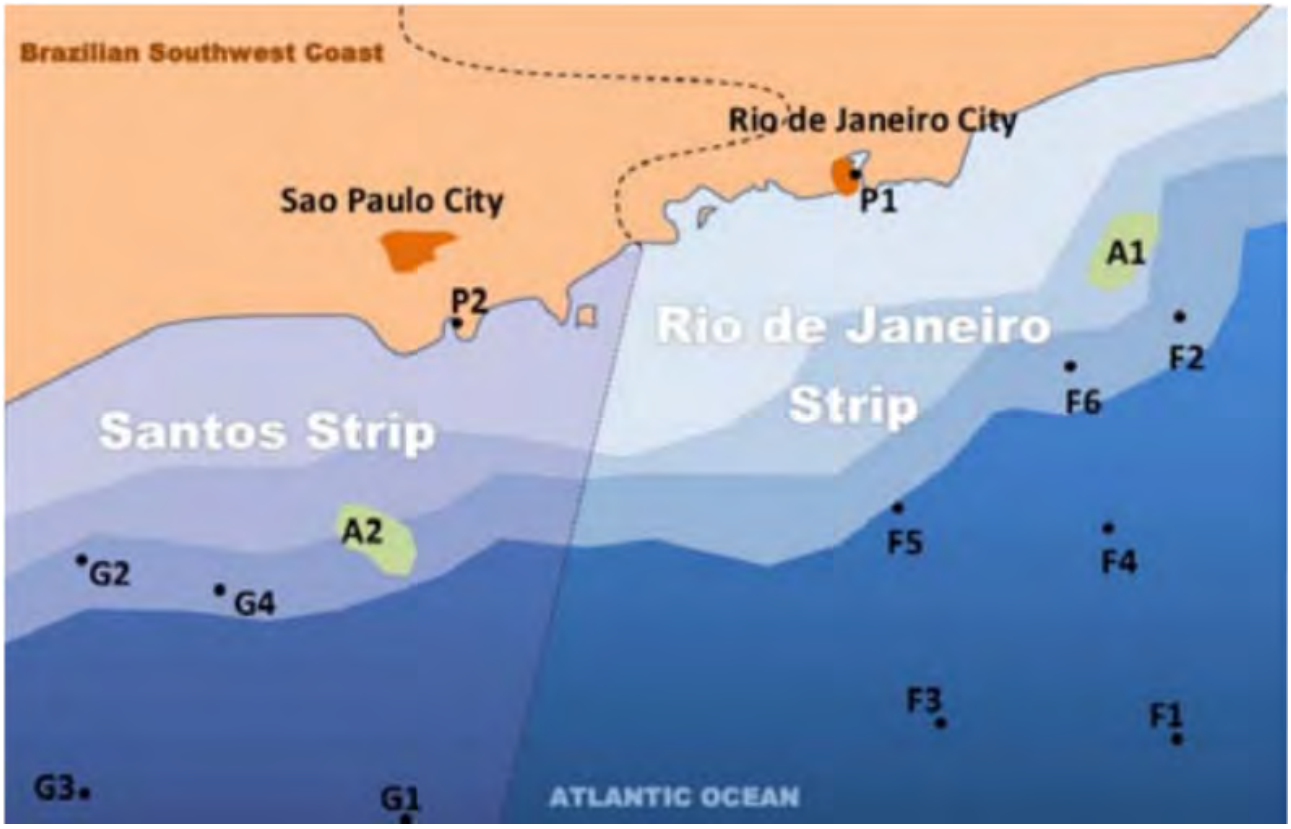

Figure 6.11 – Locations in the Petrobras domain [135].

6.3.1 Setting

Locations are static across all problems and follow the representation shown in Figure 6.11 [135]. The first problem[3] has 4 vessels (agents) and 4 cargo (goals). Each subsequent problem adds one agent and one goal, thus, the last problem has 13 vessels and 13 cargo. The configuration of each problem is shown in Table 6.4.

Table 6.4 – Petrobras problem configurations.

	vessels	**locations**	**cargo**
p01	4	14	4
p02	5	14	5
p03	6	14	6
p04	7	14	7
p05	8	14	8
p06	9	14	9
p07	10	14	10
p08	11	14	11
p09	12	14	12
p10	13	14	13

6.3.2 Results

The Petrobras domain results for time spent in planning are shown in Figure 6.12. In these graphs, the y-axis is not in logarithmic scale as in the planning time graphs from the previous two domains, since there was not a large difference in planning time as before. However, SIW+ -then-BFS(f) still appears to scale poorly with the increase in the number of agents and goals. The other planners achieve comparable results.

ADP-legacy and CMAP-t are not shown in problem 1 (p01) in all graphs of the Petrobras domain results because they had an exception while parsing it, and thus, could not find any solution for the problem.

In regards to plan size, we can observe in Figure 6.13 that DOMAP did not had the highest plan sizes as in previous domains. In the Petrobras domain, SIW+ -then-BFS(f) had the worst plan lengths across all problems.

Although SIW+ -then-BFS(f) often had good parallelism in previous domains, it manages to compete directly with DOMAP in the Petrobras domain, even surpassing DOMAP in some problems, as can be seen in Figure 6.14. On the other hand, the remaining planners performed much worse than in previous domains.

[3]The HTN encodings that we used are available at https://github.com/smart-pucrs/petrobras-domain

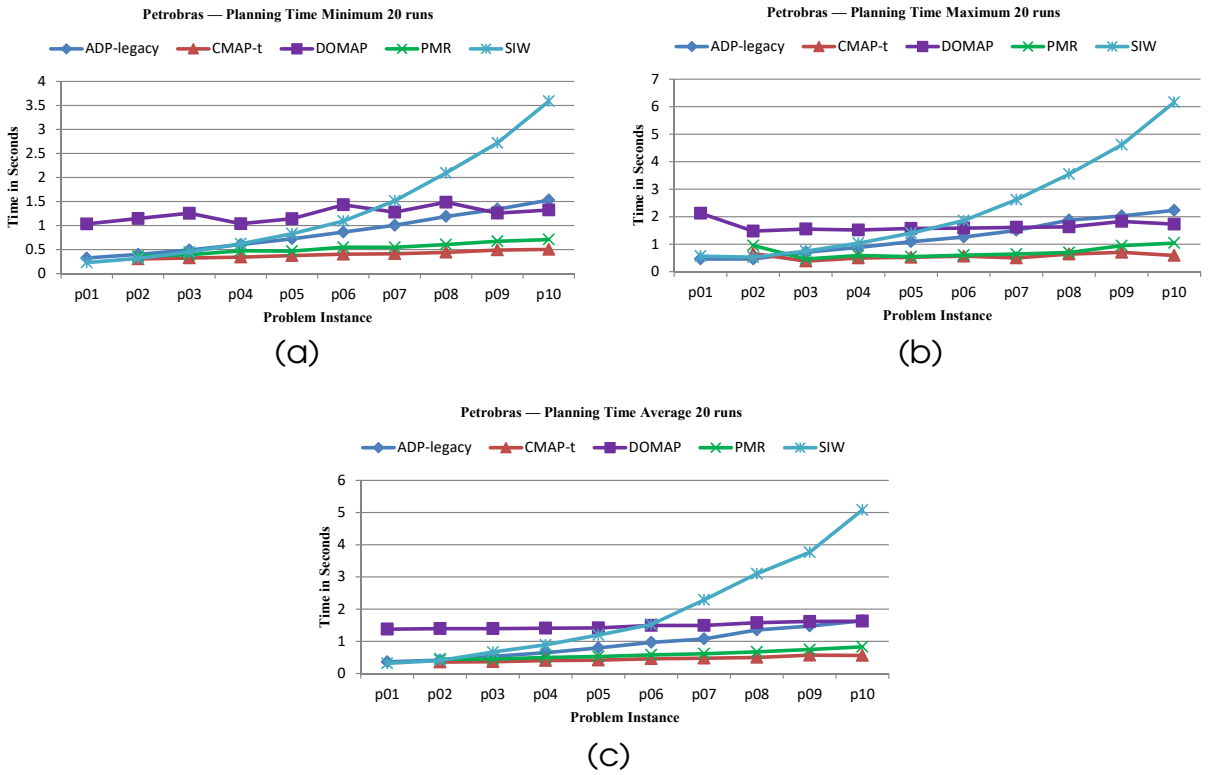

Figure 6.12 – (a) Minimum time spent planning; (b) Maximum time spent planning; (c) Average time spent planning.

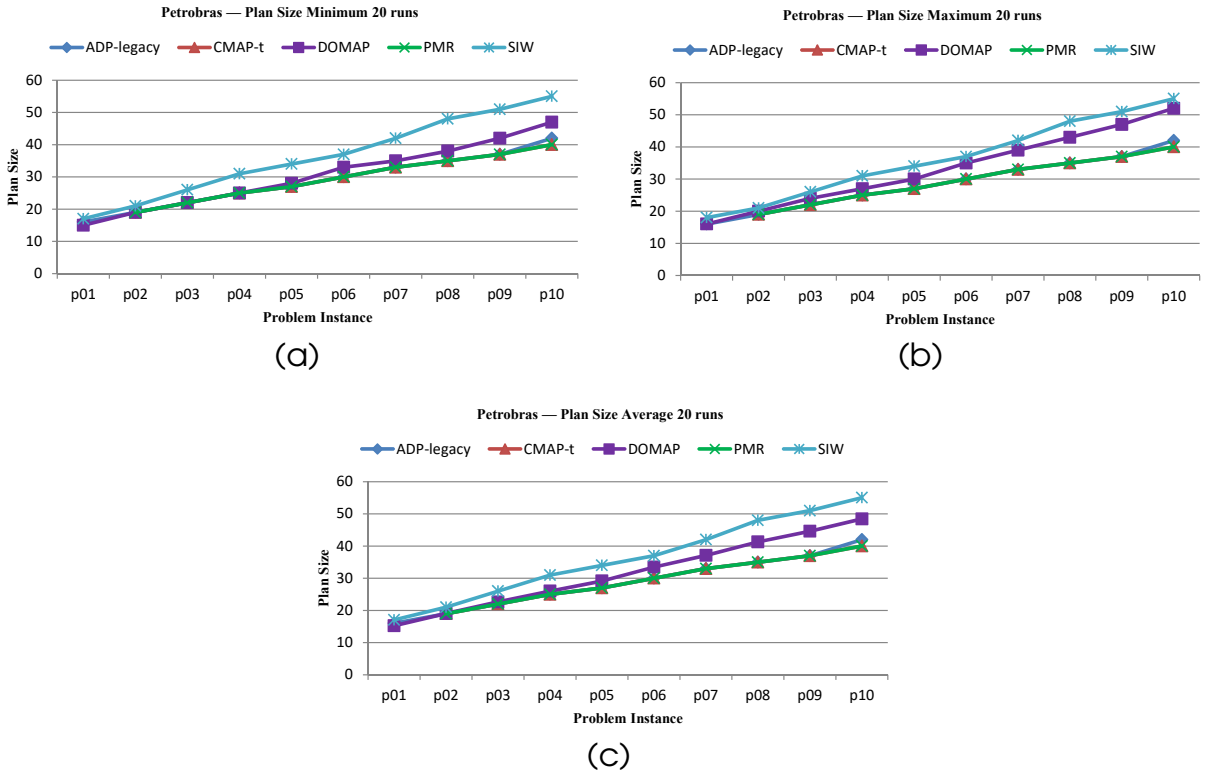

Figure 6.13 – (a) Minimum plan size; (b) Maximum plan size; (c) Average plan size.

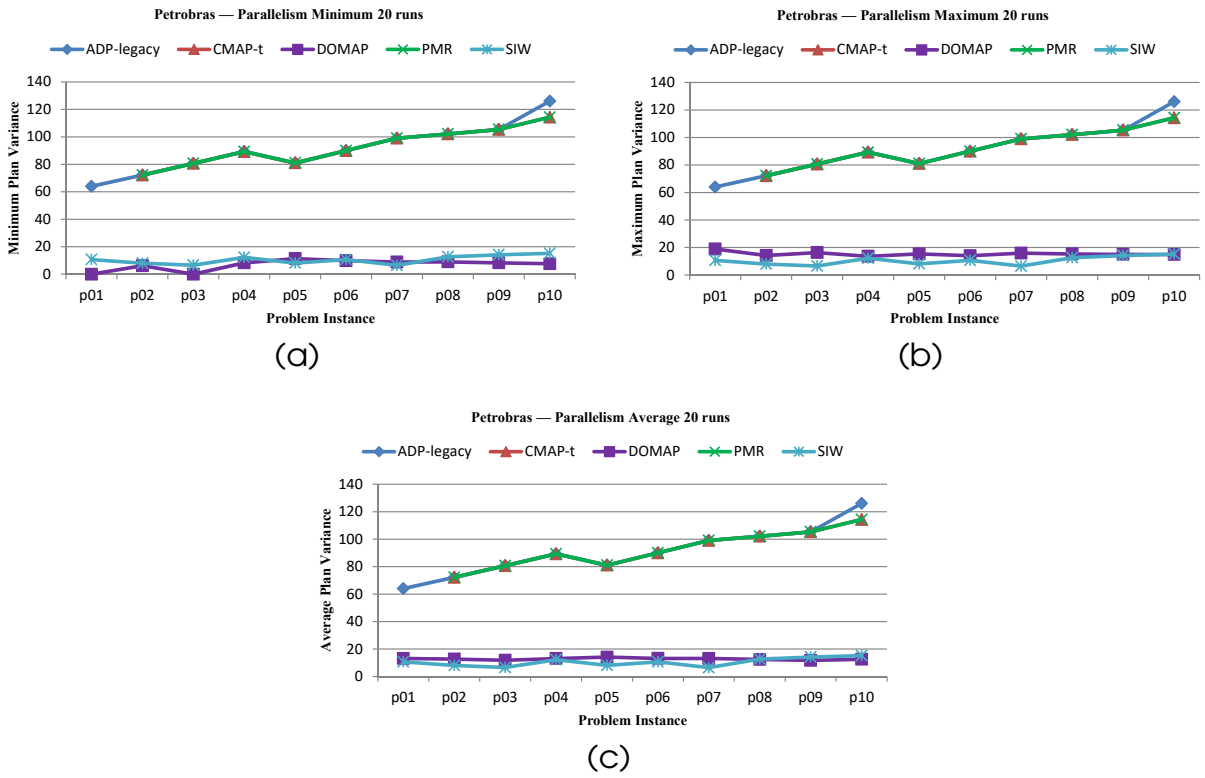

Figure 6.14 – (a) Minimum plan size variance between all agents; (b) Maximum plan size variance between all agents; (c) Average plan size variance between all agents.

The impact of parallelism on the execution's performance is even more apparent with the execution time results from Figure 6.15. SIW+ -then-BFS(f) and DOMAP, the both planners who had the best parallelism, achieve the faster execution times.

The combination of planning and execution times is shown in Figure 6.16. There was not any large differences in time (planning or execution) as in the previous domains, thus, results for all planners were included in both graphs. ADP-legacy and CMAP-t are not shown in problem 1 (p01) in Figure 6.16a because they did not succeed in planning, and consequently, did not have any execution results for this problem.

Appendix E contains tables with all values used to generate each of the graphs found in this section.

6.3.3 Discussion

All four PDDL planners use the same input language, that is, they all used the same domain and problem specifications. However, only CMAP-t and PMR (both use the same parser from MA-PDDL to PDDL with obfuscation) had parsing errors in the first problem of the Petrobras domain, and were not able to perform planning. In our experiments, this was the only instance where a planner crashed, although we suspect that SIW+ -then-BFS(f) also did something wrong in the parsing of problem 8 from the

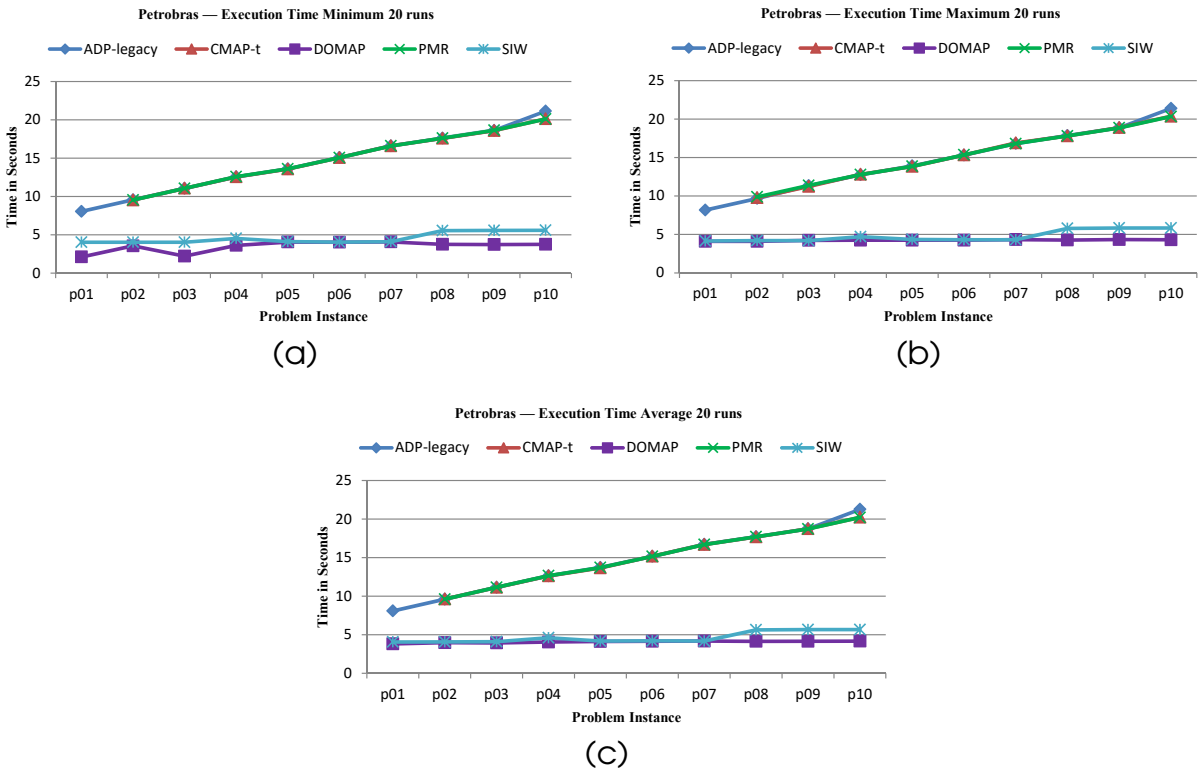

Figure 6.15 – (a) Minimum time spent executing solutions; (b) Maximum time spent executing solutions; (c) Average time spent executing solutions.

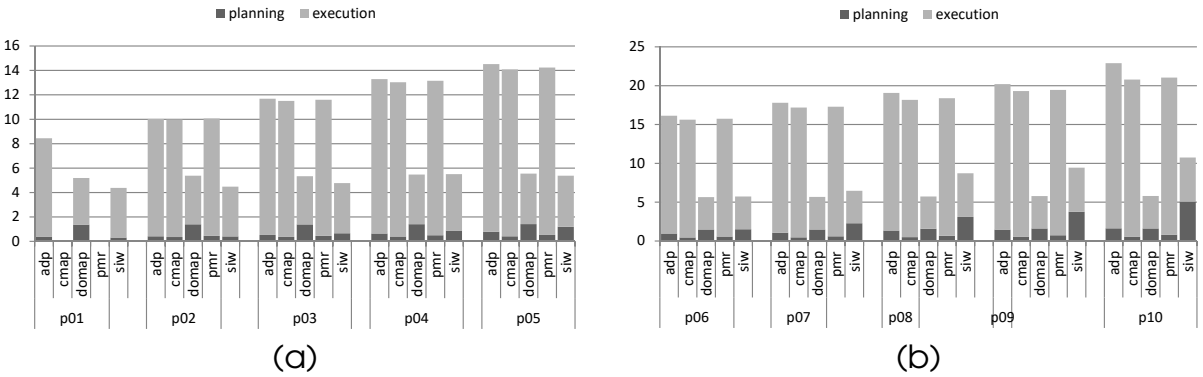

Figure 6.16 – (a) Planning and execution times for the first 5 problems in the Petrobras domain; (b) Planning and execution times for the last 5 problems in the Petrobras domain.

Rovers domain, as it was not a particularly hard problem to solve (when compared to problem 9 and 10).

Our problems in the Petrobras domain were not as difficult (in relation to the total number of agents and goals) as the previous two domains. Consequently, DOMAP maintained its base planning time of under two seconds, which we have already covered in previous domains is due to start-up of the MAS and the use of contract net protocol. These results are in line with the results from the other two domains, and we expect that larger problems would result in good scalability regarding planning time for DOMAP.

The DOMAP results for plan size are most likely due to the low number of agents in this domain, 13 in problem 10 against 22 in Rovers and 36 in Floods; and because the number of locations do not increase with each problem.

Regarding parallelism, DOMAP achieves an impressive minimum variance of zero in problem 1 and 3. This indicates the best possible goal allocation, where the tasks are distributed evenly across all agents. This is reflected in the execution results, which when combined with the planning time results, show that DOMAP maintains a modest performance in the first six problems, and outperforms all other planners in the last four problems.

7. CONCLUSION

In this thesis, we described the DOMAP framework and its four main components: (*i*) multi-agent factored representation — a multi-agent interface between agent, environment, and organisation dimensions from MAS and HTN planning; (*ii*) goal allocation mechanism — by using a contract net protocol, agents that participate in the planning phase can pre-select goals that they believe to be more appropriate for them, which can cut planning time considerably in domains with heterogeneous agents; (*iii*) individual planner — the SHOP2 planner is used in each agent for individual planning, so as to make the most of the HTN-like structure of the plan library found in typical BDI agents; (*iv*) coordination mechanism — employing social laws to coordinate agents at execution time, in order to avoid any conflicts created during planning.

We also described our implementation of DOMAP with the JaCaMo MAS development platform. Detailing how each of the different dimensions in JaCaMo interact with DOMAP's main components. In the organisation level, we demonstrated how to define social laws and created an organisation artifact responsible for ensuring that agents follow social laws when executing conflicting actions, and how the failure of social schemes can start DOMAP to generate new, and improved, social schemes. In the environment level, we described artifacts responsible for parsing information between the MAS and the individual planner, as well as CNP artifacts for goal allocation. In the agent level, we presented the set of Jason domain-independent plans to explore an agent's plan library, generating goal-plan trees, and extracting useful information for CNP bids.

7.1 Summary of Results

Experiments in the classical Rovers domain and Petrobras domain were reported, but in order to better evaluate our approach we defined a new multi-agent problem, the Floods domain. In this domain, three heterogeneous types of agents (UAVs, USVs, and UGVs) provide aid to centres for disaster management in order to monitor possible flood areas and provide any necessary assistance in regions affected by floods. We isolated the online components of DOMAP (MAS execution and social laws) and ran experiments in those domains comparing against four other state-of-the-art multi-agent planners from CoDMAP-15, and then used DOMAP MAS execution and social laws components to run the solutions found by all planners.

Our results show that DOMAP scales rather well, provides excellent parallel solutions, and is the fastest multi-agent planner for the largest problems, while still maintaining a reasonable plan length compared to other planners. Our results indicate that

allocating goals before planning can lead to a huge improvement with regards to planning time, since SIW+ -then-BFS(f) is the only planner that does not do so. Moreover, decentralising planning and running in computers with multiple cores can also lead to faster planning times (only DOMAP and PMR decentralise planning). Furthermore, our experiments show that when concurrent actions are possible, lower plan length does not directly translate into lower execution times, and that fairness can be as important as the overall plan length in some multi-agent settings. While most other multi-agent planners favour the *makespan* metric (largest plan size of agents), we have shown that our *parallelism* metric is a better indicator for execution performance, at least when there are many concurrent actions. Finally, when combining time spent on planning and execution, DOMAP achieves the best performance.

Our results with regards to planning time, contradict those found in CoDMAP-15, which we believe was mostly due to the low scale factor of the problems found in the competition. Their domains were too simplistic, with a small number of agents and goals, resulting in planning times low enough that the winner could have been any of the top performing planners. One of the advantages of MAP is its usability in large-scale problems, with many agents and goals. Such problems are still much needed for improving benchmarking in the MAP community, and in this thesis we have introduced the Floods domain as a first step towards this goal.

7.2 Future Work

The recent addition of the interaction as a programming abstraction to JaCaMo can allow further improvements to both goal allocation and the coordination mechanism used in DOMAP. In [152], an implementation of CNP as an interaction protocol is given, which could be used instead of the artifact approach we currently use, or maybe even a combination of both. As for the coordination mechanism, social laws could easily be described as interaction protocols, which might prove to be a more straightforward process then defining them as CArtAgO artifacts, which were already quite limited for this purpose.

The planners ADP-legacy, CMAP-T, and PMR all use relaxed planning graphs during goal allocation, and we believe that DOMAP's plan length can be improved considerably if we integrate relaxed planning graphs in each agent bid calculation together with our current bid selection heuristics.

One of our goals with DOMAP, is to turn it into a general-purpose domain-independent framework. As such, we designed DOMAP to be an open platform where other alternatives for modular components can be used, allowing the developer to choose the approach that is more suited for their particular problem. For example, by taking an

argumentation-based approach [113, 97] instead of using contract net protocol, agents could cooperate better, since a communication link would be established instead of just placing bids according to what they believe to be their own value. This improvement in cooperation could lead to better goal allocation, especially in tightly-coupled domains with many conflicts and dependencies.

There are many other formalisms, mechanisms, and planners that could be added to DOMAP. The following is a list of some of the alternatives that we have considered:

- **formalism:** MA-STRIPS [17], MA PDDL 3.1 [71].
- **goal allocation:** vickrey auction, market simulation and economics, MDPs, game theory.
- **individual planner:** fast downward [59], temporal planners, probabilistic planners.
- **coordination mechanism:** temporal decoupling, plan merging, simple temporal networks.

Another interesting experiment to make would be to allow the agents to choose the planner that is the most appropriate for achieving a particular goal. For example, while one agent might opt to use SHOP2 planner, another agent might have a goal that contains some degree of uncertainty, where a probabilistic planner would be better suited for the job.

Considering self-interested agents instead of fully cooperative ones opens up the use of many negotiation techniques in goal allocation. While coordination might prove to be more difficult in these instances, some sort of negotiation could also be applied. Privacy also becomes much more important, as agents need to protect their information from other agents, as each agent expects to win their negotiations.

The Multi-Agent Programming Contest scenarios [28] are very complex MAS that would make very interesting multi-agent planning domains. Although the small deadline for sending an action at each step of the simulation (currently four seconds) is a challenge for applying planning techniques, with DOMAP and the results shown here we believe it is possible, as long as a good translation between the environment and the factored representation is in place (e.g., using only relevant states for planning instead of the whole environment).

There is a possibility of running DOMAP in a real-world version of the Floods scenario (presented in Section 6.2). There are two USVs available, of the *Lutra Prop*[1] model shown in Figure 7.1, and several USVs and UGVs. The boats are low-cost, robust, and compact in size and weight. They are equipped with an electrical conductivity

[1] http://senseplatypus.com/

sensor, a dissolved oxygen sensor, and one side-scan sonar. Experiments with the boats are already under way in a separate project, and eventually JaCaMo agents should be controlling them, enabling the use of DOMAP.

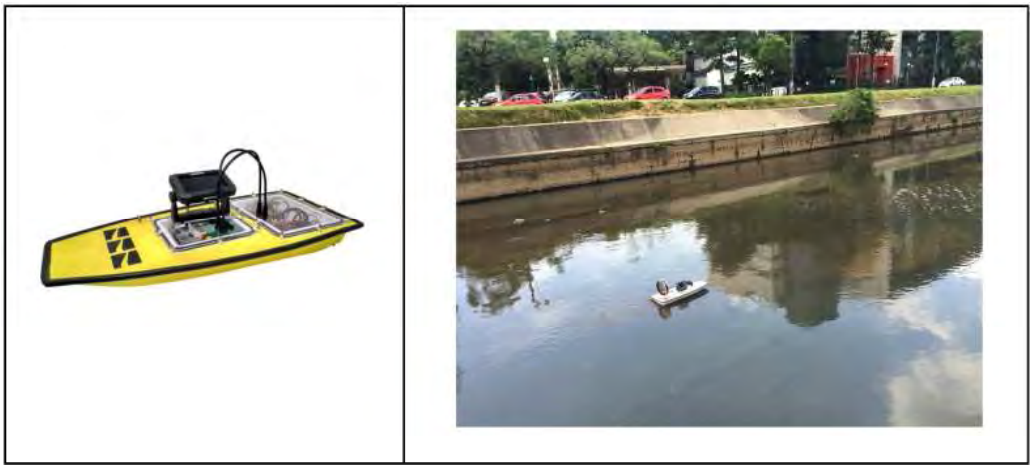

Figure 7.1 – A still image of Lutra Prop on the left, and an image of it in our first field work.

Another interesting line of future work is related to the validation of HTN plans, to check if the solution found during planning is still valid during execution. There is an algorithm [7] that uses classical parsing of context-free grammars customized to attribute grammars with a timeline constraint [5]. Experiments indicate that converting HTN models to attribute grammars may provide better time-performance results in comparison to converting to a SAT-based approach [8].

REFERENCES

[1] Alberola, J. M.; Such, J. M.; Espinosa, A.; Botti, V.; García-Fornes, A. “Magentix: a Multiagent Platform Integrated in Linux”. In: European Workshop on Multi-Agent Systems, 2008, pp. 1–10.

[2] Alechina, N.; Behrens, T.; Hindriks, K.; Logan, B. “Query caching in agent programming languages”. In: Proceedings of the 10th International Workshop on Programming Multiagent Systems, 2012, pp. 117–131.

[3] Backus, J. W. “The syntax and semantics of the proposed international algebraic language of the zurich ACM-GAMM conference”. In: International Federation for Information Processing Congress, 1959, pp. 125–131.

[4] Barrett, A.; Weld, D. S. “Partial-order planning: Evaluating possible efficiency gains”, *Artificial Intelligence*, vol. 67–1, May 1994, pp. 71–112.

[5] Barták, R.; Maillard, A. “Attribute grammars with set attributes and global constraints as a unifying framework for planning domain models”. In: Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, 2017, pp. 39–48.

[6] Barták, R.; Zhou, N.-F. “On modeling planning problems: Experience from the petrobras challenge”. In: Advances in Soft Computing and Its Applications, 2013, pp. 466–477.

[7] Barták, R.; Maillard, A.; Cardoso, R. C. “Validation of hierarchical plans via parsing of attribute grammars”. In: Proceedings of the Twenty-Eight International Conference on Automated Planning and Scheduling, 2018, 8p.

[8] Behnke, G.; Höller, D.; Biundo, S. “This is a solution! (... but is it though?) verifying solutions of hierarchical planning problems”. In: Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, 2017, pp. 20–28.

[9] Behrens, T. M.; Hindriks, K.; Hübner, J.; Dastani, M. “Putting APL platforms to the test: Agent similarity and execution performance”, Technical Report, Clausthal University of Technology, 2010, 23p.

[10] Bellifemine, F. L.; Caire, G.; Greenwood, D. “Developing Multi-Agent Systems with JADE”. John Wiley & Sons, 2007, 286p.

[11] Bellman, R. “A Markovian Decision Process”, *Indiana University Mathematics Journal*, vol. 6–4, Apr 1957, pp. 679–684.

[12] Bofill, M.; Espasa, J.; Villaret, M. “Efficient SMT encodings for the petrobras domain”. In: Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation, 2014, pp. 68–84.

[13] Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; Santi, A. “Multi-agent oriented programming with JaCaMo”, *Science of Computer Programming*, vol. 78–6, Jun 2013, pp. 747–761.

[14] Bordini, R. H.; Dix, J. “Programming multiagent systems”. In: *Multiagent Systems 2nd Edition*, Weiss, G. (Editor), MIT Press, 2013, chap. 11, pp. 587–639.

[15] Bordini, R. H.; Wooldridge, M.; Hübner, J. F. “Programming Multi-Agent Systems in AgentSpeak using Jason”. John Wiley & Sons, 2007, 273p.

[16] Borrajo, D.; Fernandez, S. “MAPR and CMAP”. In: Proceedings of the Competition of Distributed and Multi-Agent Planners, 2015, pp. 1–3.

[17] Brafman, R. I.; Domshlak, C. “From one to many: Planning for loosely coupled multi-agent systems”. In: International Conference on Automated Planning and Scheduling, 2008, pp. 28–35.

[18] Brafman, R. I.; Shani, G.; Zilberstein, S. “Qualitative planning under partial observability in multi-agent domains”. In: Proceedings of the Twenty-Seventh Conference on Artificial Intelligence, 2013, pp. 130–137.

[19] Bratman, M. E.; Israel, D. J.; Pollack, M. E. “Plans and resource-bounded practical reasoning”, *Computational Intelligence*, vol. 4–3, Sep 1988, pp. 349–355.

[20] Braubach, L.; Pokahr, A.; Lamersdorf, W. “A universal criteria catalog for evaluation of heterogeneous agent development artifacts”. In: From Agent Theory to Agent Implementation, 2008, pp. 19–28.

[21] Brenner, M.; Nebel, B. “Continual planning and acting in dynamic multiagent environments”, *Autonomous Agents and Multi-Agent Systems*, vol. 19–3, Dec 2009, pp. 297–331.

[22] Cardoso, R. C.; Bordini, R. H. “Allocating social goals using the contract net protocol in online multi-agent planning”. In: 5th Brazilian Conference on Intelligent System, 2016, pp. 199–204.

[23] Cardoso, R. C.; Bordini, R. H. “A distributed online multi-agent planning system”. In: 4th Workshop on Distributed and Multi-Agent Planning, 2016, pp. 15–23.

[24] Cardoso, R. C.; Bordini, R. H. “A modular framework for decentralised multi-agent planning”. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 1487–1489.

[25] Cardoso, R. C.; Bordini, R. H. “A multi-agent extension of a hierarchical task network planning formalism”, *Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 6–2, May 2017, pp. 5–17.

[26] Cardoso, R. C.; Hübner, J. F.; Bordini, R. H. “Benchmarking communication in actor- and agent-based languages”. In: 12th International Conference on Autonomous Agents and Multiagent Systems, 2013, pp. 1267–1268.

[27] Cardoso, R. C.; Hübner, J. F.; Bordini, R. H. “Benchmarking communication in agent- and actor-based languages”. In: Engineering Multi-Agent Systems, 2013, pp. 81–96.

[28] Cardoso, R. C.; Pereira, R. F.; Krzisch, G.; Magnaguagno, M. C.; Basegio, T.; Meneguzzi, F. “Team pucrs: a decentralised multi-agent solution for the agents in the city scenario”, *International Journal of Agent-Oriented Software Engineering*, vol. 6–1, Jan 2018, pp. 3–34.

[29] Cardoso, R. C.; Zatelli, M. R.; Hübner, J. F.; Bordini, R. H. “Towards benchmarking actor- and agent-based programming languages”. In: Workshop on Programming based on actors, agents, and decentralized control, 2013, pp. 115–126.

[30] Clearwater, S. H. (Editor). “Market-based Control: A Paradigm for Distributed Resource Allocation”. World Scientific Publishing Co. Inc., 1996, 328p.

[31] Clement, B. J.; Durfee, E. H.; Barrett, A. C. “Abstract reasoning for planning and coordination”, *Journal of Artificial Intelligence Research*, vol. 28–1, Apr 2007, pp. 453–515.

[32] Coen, M. H. “Non-deterministic social laws”. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, 2000, pp. 15–21.

[33] Collier, R. W.; Russell, S.; Lillis, D. “Reflecting on agent programming with AgentSpeak(L)”. In: Principles and Practice of Multi-Agent Systems, 2015, pp. 351–366.

[34] Cox, J.; Durfee, E. “Efficient and distributable methods for solving the multiagent plan coordination problem”, *Multiagent and Grid Systems*, vol. 5–4, Dec 2009, pp. 373–408.

[35] Crosby, M. “ADP an agent decomposition planner codmap 2015”. In: Proceedings of the Competition of Distributed and Multi-Agent Planners, 2015, pp. 4–7.

[36] Crosby, M.; Jonsson, A.; Rovatsos, M. “A single-agent approach to multiagent planning”. In: 21st European Conference on Artificial Intelligence, 2014, pp. 237–242.

[37] Crosby, M.; Rovatsos, M.; Petrick, R. P. A. “Automated agent decomposition for classical planning”. In: Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, 2013, pp. 46–54.

[38] Dastani, M. “2APL: a practical agent programming language”, *Autonomous Agents and Multi-Agent Systems*, vol. 16–3, Jun 2008, pp. 214–248.

[39] de Silva, L. “BDI agent reasoning with guidance from HTN recipes”. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 759–767.

[40] de Silva, L.; Sardiña, S.; Padgham, L. “First principles planning in BDI systems”. In: 8th International Joint Conference on Autonomous Agents and Multiagent Systems, 2009, pp. 1105–1112.

[41] de Silva, L.; Sardiña, S.; Padgham, L. “Summary information for reasoning about hierarchical plans”. In: European Conference on Artificial Intelligence, 2016, pp. 1300–1308.

[42] de Weerdt, M. “Plan merging in multi-agent systems”, Ph.D. Thesis, Delft University of Technology, 2003, 204p.

[43] de Weerdt, M.; Clement, B. “Introduction to planning in multiagent systems”, *Multiagent and Grid Systems*, vol. 5–4, Dec 2009, pp. 345–355.

[44] Dean, T.; Boddy, M. S. “An analysis of time-dependent planning.” In: National Conference on Artificial Intelligence, 1988, pp. 49–54.

[45] Decker, K. S. “TAEMS: A framework for environment centered analysis & design of coordination mechanisms”. In: Foundations of Distributed Artificial Intelligence, 1996, pp. 429–448.

[46] desJardins, M. E.; Durfee, E. H.; Ortiz, C. L.; Wolverton, M. J. “A survey of research in distributed, continual planning”, *AI Magazine*, vol. 20–4, Dec 1999, pp. 13–22.

[47] Díaz, Á. F.; Earle, C. B.; Fredlund, L.-A. “eJason: an implementation of Jason in Erlang”. In: Proceedings of the 10th International Workshop on Programming Multi-Agent Systems, 2012, pp. 7–22.

[48] Dignum, V.; Padget, J. “Multiagent organizations”. In: *Multiagent Systems 2nd Edition*, Weiss, G. (Editor), MIT Press, 2013, chap. 2, pp. 51–98.

[49] Durfee, E. H. “Distributed problem solving and planning”. In: *Mutli-agents Systems and Applications*, Carbonell, J. G.; Siekmann, J. (Editors), Springer-Verlag New York Inc., 2001, pp. 118–149.

[50] Durfee, E. H.; Zilberstein, S. “Multiagent planning, control, and execution”. In: *Multiagent Systems 2nd Edition*, Weiss, G. (Editor), MIT Press, 2013, chap. 11, pp. 485–545.

[51] Ferber, J.; Gutknecht, O.; Michel, F. “From agents to organizations: An organizational view of multi-agent systems”. In: Agent-Oriented Software Engineering, LNCS 2935, 2003, pp. 214–230.

[52] Fernández, V.; Grimaldo, F.; Lozano, M.; Orduña, J. M. “Evaluating Jason for distributed crowd simulations”. In: International Conference on Agents and Artificial Intelligence., 2010, pp. 206–211.

[53] Fikes, R. E.; Nilsson, N. J. “STRIPS: a new approach to the application of theorem proving to problem solving”. In: Proceedings of the 2nd international joint conference on Artificial intelligence, 1971, pp. 608–620.

[54] Fitoussi, D.; Tennenholtz, M. “Choosing social laws for multi-agent systems: Minimality and simplicity”, *Artificial Intelligence*, vol. 119–1, Aug 2000, pp. 61–101.

[55] Foulser, D. E.; Li, M.; Yang, Q. “Theory and algorithms for plan merging”, *Artificial Intelligence*, vol. 57–2, Oct 1992, pp. 143–181.

[56] Georgeff, M.; Lansky, A. “Procedural knowledge”, *Proceedings of the Institute of Electrical and Electronics Engineers (Special Issue on Knowledge Representation)*, vol. 74–10, Oct 1986, pp. 1383–1398.

[57] Giacomo, G.; Lespérance, Y.; Levesque, H. J.; Sardina, S. “Indigolog: A high-level programming language for embedded reasoning agents”. In: *Multi-Agent Programming: Languages, Tools and Applications*, Bordini, R. H.; Dastani, M.; Dix, J.; Seghrouchni, A. E. F. (Editors), Springer, 2009, chap. 2, pp. 31–72.

[58] Gmytrasiewicz, P. J.; Durfee, E. H. “Rational coordination in multi-agent environments”, *Autonomous Agents and Multi-Agent Systems*, vol. 3–4, Dec 2000, pp. 319–350.

[59] Helmert, M. “The fast downward planning system”, *Journal of Artificial Intelligence Research*, vol. 26–1, Jul 2006, pp. 191–246.

[60] Hindriks, K. V.; de Boer, F. S.; van der Hoek, W.; Meyer, J.-J. C. “Agent programming with declarative goals”. In: Proceedings of the 7th International Workshop on Agent Theories and Architectures„ 2000, pp. 228–243.

[61] Hindriks, K. V.; Roberti, T. “Goal as a planning formalism”. In: Proceedings of Multiagent System Technologies, Braubach, L.; van der Hoek, W.; Petta, P.; Pokahr, A. (Editors), 2009, pp. 29–40.

[62] Hoek, W.; Roberts, M.; Wooldridge, M. “Social laws in alternating time: effectiveness, feasibility, and synthesis”, *Synthese*, vol. 156–1, May 2006, pp. 1–19.

[63] Hoffmann, J.; Nebel, B. “The FF planning system: fast plan generation through heuristic search”, *Journal of Artificial Intelligence Research*, vol. 14–1, May 2001, pp. 253–302.

[64] Howden, N.; Rönnquist, R.; Hodgson, A.; Lucas, A. “JACK intelligent agents - summary of an agent infrastructure”. In: 5th International conference on autonomous agents, 2001, 6p.

[65] Hübner, J. F.; Sichman, J. S.; Boissier, O. “Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels”, *International Journal of Agent-Oriented Software Engineering*, vol. 1–3, 2007, pp. 370–395.

[66] Igreja, H.; Silva, J. R.; Tonidandel, F. “ICKEPS 2012 challenge domain: Planning ship operations on petroleum platforms and ports”. In: Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling, 2012, 11p.

[67] Jordan, H.; Botterweck, G.; Huget, M.-P.; Collier, R. “A feature model of actor, agent, and object programming languages”. In: Workshop on Programming based on actors, agents, and decentralized control, 2011, pp. 147–158.

[68] Jr., J. C. B.; Durfee, E. H. “Distributed algorithms for solving the multiagent temporal decoupling problem”. In: International Conference on Autonomous Agents and Multiagent Systems, 2011, pp. 141–148.

[69] Kambhampati, S. “Refinement planning as a unifying framework for plan synthesis”, *AI Magazine*, vol. 18–2, Jun 1997, pp. 67–97.

[70] Komenda, A.; Novak, P.; Pechoucek, M. “Domain-independent multi-agent plan repair”, *Journal of Network and Computer Applications*, vol. 37, Jan 2014, pp. 76–88.

[71] Kovacs, D. L. “A multi-agent extension of PDDL 3.1”. In: Proceedings of the 3rd Workshop on the International Planning Competition, 2012, pp. 19–27.

[72] Kuter, U.; Nau, D. S. “Using domain-configurable search control for probabilistic planning”. In: Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, 2005, pp. 1169–1174.

[73] Kuter, U.; Nau, D. S.; Pistore, M.; Traverso, P. “A hierarchical task-network planner based on symbolic model checking”. In: Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, 2005, pp. 300–309.

[74] Lerma, A. T. “Design and implementation of a multi-agent planning system”, Master’s Thesis, Polytechnic University of Valencia, 2011, 81p.

[75] Levesque, H.; Reiter, R. “High-level robotic control: Beyond planning. a position paper”. In: Association for the Advancement of Artificial Intelligence 1998 Spring Symposium: Integrating Robotics Research: Taking the Next Big Leap, 1998, pp. 106–108.

[76] Levesque, H. J.; Reiter, R.; Lin, F.; Scherl, R. B. “Golog: A logic programming language for dynamic domains”, *Journal of Logic Programming*, vol. 31–1, Apr 1997, pp. 54–84.

[77] Luis, N.; Borrajo, D. “PMR: Plan merging by reuse”. In: Proceedings of the Competition of Distributed and Multi-Agent Planners, 2015, pp. 11–13.

[78] Mao, X.; Mors, A.; Roos, N.; Witteveen, C. “Coordinating competitive agents in dynamic airport resource scheduling”. In: Proceedings of the 5th German conference on Multiagent System Technologies, 2007, pp. 133–144.

[79] Mcdermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; Wilkins, D. “PDDL - the planning domain definition language”, Technical Report, Yale Center for Computational Vision and Control, 1998, 27p.

[80] Meneguzzi, F.; De Silva, L. “Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning”, *The Knowledge Engineering Review*, vol. 30–1, Jan 2015, pp. 1–44.

[81] Meneguzzi, F.; Luck, M. “Composing high-level plans for declarative agent programming”. In: *Declarative Agent Languages and Technologies V*, Baldoni, M.; Son, T.; van Riemsdijk, M.; Winikoff, M. (Editors), Springer Berlin Heidelberg, 2008, pp. 69–85.

[82] Meneguzzi, F. R.; Zorzo, A. F.; Móra, M. D. C. “Propositional planning in BDI agents”. In: Proceedings of the 2004 ACM Symposium on Applied Computing, 2004, pp. 58–63.

[83] Miller, G. E. “Hubble space telescope planning and scheduling - experience; lessons learned and future directions”. In: Proceedings of the workshop New observing modes for the next century, 1996, pp. 158–161.

[84] Mora, M. C.; Lopes, J. G.; Viccariz, R. M.; Coelho, H. “BDI models and systems: Reducing the gap”. In: Intelligent Agents V: Agents Theories, Architectures, and Languages, 1999, pp. 11–27.

[85] Muise, C.; Lipovetzky, N.; Ramirez, M. “MAP-LAPKT: Omnipotent multi-agent planning via compilation to classical planning”. In: Proceedings of the Competition of Distributed and Multi-Agent Planners, 2015, pp. 14–16.

[86] Murphy, R. R. “Trial by fire [rescue robots]”, *IEEE Robotics Automation Magazine*, vol. 11–3, Sep 2004, pp. 50–61.

[87] Murphy, R. R. “Emergency informatics: Using computing to improve disaster management”, *Computer*, vol. 49–5, May 2016, pp. 19–27.

[88] Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Wu, D.; Yaman, F.; Munoz-Avila, H.; Murdock, J. “Applications of SHOP and SHOP2”, *Intelligent Systems*, vol. 20–2, March 2005, pp. 34–41.

[89] Nau, D.; Ghallab, M.; Traverso, P. “Automated Planning: Theory & Practice”. Morgan Kaufmann Publishers Inc., 2004, 638p.

[90] Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; Yaman, F. “SHOP2: An HTN planning system”, *Journal of Artificial Intelligence Research*, vol. 20–1, Dec 2003, pp. 379–404.

[91] Nau, D. S.; Ghallab, M.; Traverso, P. “Blended planning and acting: Preliminary approach, research challenges”. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4047–4051.

[92] Nissim, R.; Brafman, R. I. “Multi-agent A* for parallel and distributed systems”. In: Proceedings of the Heuristics for Domain-Independent Planning Workshop, 2012, pp. 1265–1266.

[93] Nissim, R.; Brafman, R. I. “Distributed heuristic forward search for multi-agent planning”, *Journal of Artificial Intelligence Research*, vol. 51–1, Oct 2014, pp. 293–332.

[94] Nissim, R.; Brafman, R. I.; Domshlak, C. “A general, fully distributed multi-agent planning algorithm”. In: International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 1323–1330.

[95] O’Hare, G. M. P. “Agent factory: an environment for the fabrication of multiagent systems”. In: Foundations of distributed artificial intelligence, 1996, pp. 449–484.

[96] Omicini, A.; Ricci, A.; Viroli, M. “Artifacts in the A&A meta-model for multi-agent systems”, *Autonomous Agents and Multi-Agent Systems*, vol. 17–3, Dec 2008, pp. 432–456.

[97] Panisson, A. R.; Freitas, A.; Schmidt, D.; Hilgert, L.; Meneguzzi, F.; Vieira, R.; Bordini, R. H. “Arguing about task reallocation using ontological information in multi-agent systems”. In: 12th International Workshop on Argumentation in Multiagent Systems, 2015, 16p.

[98] Planken, L.; de Weerdt, M.; Witteveen, C. “Optimal temporal decoupling in multiagent systems”. In: International Conference on Autonomous Agents and Multiagent Systems, 2010, pp. 789–796.

[99] Pokahr, A.; Braubach, L.; Lamersdorf, W. “Jadex: Implementing a BDI-infrastructure for JADE agents”, *EXP - in search of innovation (Special Issue on JADE)*, vol. 3–3, Dec 2003, pp. 76–85.

[100] Rao, A. S. “AgentSpeak(L): BDI agents speak out in a logical computable language”. In: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world, 1996, pp. 42–55.

[101] Rao, A. S.; Georgeff, M. P. “BDI agents: From theory to practice”. In: Proceedings of the first International Conference on Multi-Agent Systems, 1995, pp. 312–319.

[102] Ricci, A. “From actor event-loop to agent control-loop: Impact on programming”. In: Workshop on Programming based on actors, agents, and decentralized control, 2014, pp. 121–132.

[103] Ricci, A.; Piunti, M.; Viroli, M. “Environment programming in multi-agent systems – an artifact-based perspective”, *Autonomous Agents and Multi-Agent Systems*, vol. 23–2, Sep 2011, pp. 158–192.

[104] Ricci, A.; Piunti, M.; Viroli, M.; Omicini, A. “Environment programming in CArtAgO”. In: *Multi-Agent Programming: Languages, Tools and Applications*, Bordini, R. H.; Dastani, M.; Dix, J.; Seghrouchni, A. E. F. (Editors), Springer, 2009, chap. 8, pp. 259–288.

[105] Rodriguez, S.; Gaud, N.; Galland, S. “Sarl: A general-purpose agent-oriented programming language”. In: Web Intelligence and Intelligent Agent Technologies, 2014, pp. 103–110.

[106] Russell, S. J.; Norvig, P. “Artificial Intelligence: A Modern Approach”. Prentice Hall, 2009, 1152p.

[107] Sandholm, T. W.; Lesser, V. R. “Coalitions among computationally bounded agents”, *Artificial Intelligence*, vol. 94–1, Jul 1997, pp. 99–137.

[108] Sapena, O.; Onaindia, E.; Torreño, A. “FLAP: applying least-commitment in forward-chaining planning”, *AI Communications*, vol. 28–1, Jan 2015, pp. 5–20.

[109] Sardiña, S.; de Silva, L.; Padgham, L. “Hierarchical planning in BDI agent programming languages: a formal approach”. In: 5th International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, pp. 1001–1008.

[110] Sardiña, S.; Padgham, L. “Goals in the context of BDI plan failure and planning”. In: International Conference on Autonomous Agents and Multiagent Systems, 2007, 8p.

[111] Sardiña, S.; Padgham, L. “A BDI agent programming language with failure handling, declarative goals, and planning”, *Autonomous Agents and Multi-Agent Systems*, vol. 23–1, Jul 2011, pp. 18–70.

[112] Scerri, P.; Xu, Y.; Liao, E.; Lai, J.; Sycara, K. “Scaling teamwork to very large teams”. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004, pp. 888–895.

[113] Sergio Pajares Ferrando, E. O. “Defeasible argumentation for multi-agent planning in ambient intelligence applications”. In: 11th International Conference on Autonomous Agents and Multiagent Systems, 2012, pp. 509–516.

[114] Shehory, O.; Kraus, S. “Methods for task allocation via agent coalition formation”, *Artificial Intelligence*, vol. 101–1-2, May 1998, pp. 165–200.

[115] Shoham, Y. “Agent-oriented programming”, *Artificial Intelligence*, vol. 60–1, Mar 1993, pp. 51–92.

[116] Shoham, Y.; Tennenholtz, M. “On the synthesis of useful social laws for artificial agent societies (preliminary report)”. In: Proceedings of National Conference on Artificial Intelligence, 1992, pp. 276–281.

[117] Shoham, Y.; Tennenholtz, M. “On social laws for artificial agent societies: Off-line design”, *Artificial Intelligence*, vol. 73–1-2, Feb 1995, pp. 231–252.

[118] Singh, M. P.; Chopra, A. K. “Programming multiagent systems without programming agents”. In: Programming Multi-Agent Systems, 2010, pp. 1–14.

[119] Smallwood, R. D.; Sondik, E. J. “The optimal control of partially observable markov processes over a finite horizon”, *Operations Research*, vol. 21–5, Oct 1973, pp. 1071–1088.

[120] Smith, R. G. “The contract net protocol: High-level communication and control in a distributed problem solver”, *IEEE Transactions on Computers*, vol. 29–12, Dec 1980, pp. 1104–1113.

[121] Such, J. M.; García-Fornes, A.; Espinosa, A.; Bellver, J. “Magentix2: a privacy-enhancing agent platform”, *Engineering Applications of Artificial Intelligence*, vol. 26–1, Dec 2012, pp. 96–109.

[122] Sycara, K. P. “Multiagent systems”, *AI Magazine*, vol. 19–2, Jun 1998, pp. 79–92.

[123] Sycara, K. P.; Pannu, A. “The RETSINA multiagent system: Towards integrating planning, execution and information gathering”. In: Proceedings of the second international conference on Autonomous agents, 1998, pp. 350–351.

[124] Tang, P.; Wang, H.; Qi, C.; Wang, J. “Anytime heuristic search in temporal htn planning for developing incident action plans”, *AI Communications*, vol. 25–4, Oct 2012, pp. 321–342.

[125] Tang, Y.; Meneguzzi, F.; Parsons, S.; Sycara, K. “Probabilistic hierarchical planning over MDPs”. In: Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems, 2011, pp. 1143–1144.

[126] Tang, Y.; Meneguzzi, F.; Parsons, S.; Sycara, K. “Planning over MDPs through probabilistic HTNs”. In: Proceedings of the AAAI-11 Workshop on Generalized Planning, 2011, 8p.

[127] Thangarajah, J.; Padgham, L.; Winikoff, M. “Detecting and avoiding interference between goals in intelligent agents”. In: International Joint Conference on Artificial Intelligence, 2003, pp. 721–726.

[128] Toropila, D.; Dvorak, F.; Trunda, O.; Hanes, M.; Bartak, R. “Three approaches to solve the petrobras challenge: Exploiting planning techniques for solving real-life logistics problems”. In: International Conference on Tools with Artificial Intelligence, 2012, pp. 191–198.

[129] Torreno, A.; Onaindia, E.; Sapena, O. “An approach to multiagent planning with incomplete information”. In: 20th European Conference on Artificial Intelligence, 2012, pp. 762–767.

[130] Torreño, A.; Onaindia, E.; Sapena, O. “A flexible coupling approach to multi-agent planning under incomplete information”, *Knowledge and Information Systems*, vol. 38–1, Jan 2014, pp. 141–178.

[131] Torreño, A.; Onaindia, E.; Sapena, O. “FMAP: distributed cooperative multi-agent planning”, *Applied Intelligence*, vol. 41–2, Sep 2014, pp. 606–626.

[132] Van Dyke ParunaK, H.; Brueckner, S.; Sauter, J. “Digital pheromone mechanisms for coordination of unmanned vehicles”. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, 2002, pp. 449–450.

[133] van Leeuwen, P.; Witteveen, C. “Temporal decoupling and determining resource needs of autonomous agents in the airport turnaround process”. In: Proceedings of the International Conference on Intelligent Agent Technology, 2009, pp. 185–192.

[134] Vaquero, T.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; Beck, J. C. “From requirements and analysis to PDDL in itSIMPLE3. 0”. In: Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling, 2009, pp. 54–61.

[135] Vaquero, T. S.; Costa, G.; Tonidandel, F.; Igreja, H.; Silva, J. R.; Beck, J. C. “Planning and scheduling ship operations on petroleum ports and platforms”. In: Proceedings of the Scheduling and Planning Applications woRKshop, 2012, pp. 8–16.

[136] Vickrey, W. “Counterspeculation, auctions and competitive sealed tenders”, *Journal of Finance*, vol. 16–1, 1961, pp. 8–37.

[137] Vieira, R.; Moreira, Á. F.; Wooldridge, M.; Bordini, R. H. “On the formal semantics of speech-act based communication in an agent-oriented programming language”, *Journal of Artificial Intelligence Research*, vol. 29–1, May 2007, pp. 221–267.

[138] Štolba, M.; Komenda, A.; Kovacs, D. L. “Competition of distributed and multiagent planners (CoDMAP)”. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 4343–4345.

[139] Walsh, W. E.; Wellman, M. P. “A market protocol for decentralized task allocation”. In: Proceedings of the Third International Conference on Multiagent Systems, 1998, pp. 325–332.

[140] Wellman, M. P. “A market-oriented programming environment and its application to distributed multicommodity flow problems”, *Journal of Artificial Intelligence Research*, vol. 1–1, Aug 1993, pp. 1–23.

[141] Wellman, M. P.; Greenwald, A.; Stone, P.; Wurman, P. R. “The 2001 trading agent competition”, *IEEE Internet Computing*, vol. 13–1, Dec 2000, pp. 4–12.

[142] Weyns, D.; Omicini, A.; Odell, J. “Environment as a first class abstraction in multiagent systems”, *Autonomous Agents and Multi-Agent Systems*, vol. 14–1, Feb 2007, pp. 5–30.

[143] Witwicki, S. J.; Durfee, E. H. “Towards a unifying characterization for quantifying weak coupling in dec-POMDPs”. In: International Conference on Autonomous Agents and Multiagent Systems, 2011, pp. 29–36.

[144] Wooldridge, M. “An Introduction to MultiAgent Systems”. John Wiley & Sons, 2002, 368p.

[145] Wooldridge, M. “Intelligent agents”. In: *Multiagent Systems 2nd Edition*, Weiss, G. (Editor), MIT Press, 2013, chap. 1, pp. 3–50.

[146] Wu, F.; Zilberstein, S.; Chen, X. “Online planning for multi-agent systems with bounded communication”, *Artificial Intelligence*, vol. 175–2, Feb 2011, pp. 487–511.

[147] Xueguang, C.; Haigang, S. “Further extensions of FIPA contract net protocol: Threshold plus DoA”. In: Proceedings of the 2004 ACM Symposium on Applied Computing, 2004, pp. 45–51.

[148] Yadati, C.; Witteveen, C.; Zhang, Y. “Coordinating agents - an analysis of coordination in supply-chain management tasks”. In: Proceedings of the 2nd International Conference on Agents and Artificial Intelligence, 2010, pp. 218–223.

[149] Yang, Q.; Nau, D. S.; Hendler, J. “Merging separately generated plans with restricted interactions”, *Computational Intelligence*, vol. 8–4, Feb 1992, pp. 648–676.

[150] Yokoo, M.; Durfee, E. H.; Ishida, T.; Kuwabara, K. “The distributed constraint satisfaction problem: Formalization and algorithms”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 10–5, Sep 1998, pp. 673–685.

[151] Zatelli, M. R.; de Brito, M.; Schmitz, T. L.; Morato, M. M.; de Souza, K. S.; Uez, D. M.; Hübner, J. F. “SMADAS: A team for mapc considering the organization and the environment as first-class abstractions”. In: Engineering Multi-Agent Systems, 2013, pp. 319–328.

[152] Zatelli, M. R.; Hübner, J. F. “The interaction as an integration component for the JaCaMo platform”. In: Engineering Multi-Agent Systems, 2014, pp. 431–450.

APPENDIX A – REMAINING DESCRIPTIONS OF THE FLOODS DOMAIN

The domain for an UGV is shown in Listing A.1. The navigation operator and methods are similar to UAVs, except that it has the precondition that requires a ground path between two areas, and the method has a third branch to traverse through middle areas in order to arrive at its destination (using internal operators, discussed in the previous chapter). There are two new operators: `pickup_box` to pick a first aid kit that is located in a CDM, and `drop_box` to drop it in an area that requires it. The `deliver_box` method's task list has the following recipe: navigate to the area where the box is located, pick it up, navigate to the goal area, and drop the box.

The USV domain specification in Listing A.2 is very similar to UGVs. One of the differences is that USVs require water instead of ground paths. The operators and methods related to box are swapped for water sampling ones. The `sample_water` operator collects a water sample in the area where the USV is in, while `drop_sample` drops the water sample if the agent is at a CDM. The `deliver_sample` method's recipe is to navigate to the area where a water sample was requested, collect it, move to an area with a CDM, and drop the sample.

Listing A.1 – HTN domain specification of an UGV agent.

```
(defdomain floods-ugv (
(:operator (!navigate ?from ?to)
      ( (area ?from) (area ?to) (ground_path ?from ?to) (at ?from) )
      ( (at ?from) )
      ( (at ?to) )
)
(:operator (!take_picture ?area ?disaster)
      ( (area ?area) (disaster ?disaster) (visible_from ?disaster ?area) (at ?area) )
      ( )
      ( (have_picture ?disaster) )
)
(:operator (!communicate_data ?cdm ?disaster ?area1 ?area2)
      ( (disaster ?disaster) (cdm ?cdm) (at ?area1) (cdm_at ?cdm ?area2) (area ?area1) (area ?area2) (
      have_picture ?disaster) (in_range ?area1 ?area2) )
      ( (have_picture ?disaster) )
      ( (communicated_data ?disaster) )
)
(:operator (!pickup_box ?store ?cdm ?area ?box)
      ( (store ?store) (empty ?store) (cdm ?cdm) (box ?box) (area ?area) (box_at_cdm ?box ?cdm) (
      cdm_at ?cdm ?area) (at ?area) )
      ( (empty ?store) (box_at_cdm ?box ?cdm) )
      ( (full ?store) (have_box ?box) )
)
(:operator (!drop_box ?store ?area ?box)
      ( (store ?store) (full ?store) (box ?box) (area ?area) (have_box ?box) (at ?area) )
      ( (full ?store) (have_box ?box) )
      ( (empty ?store) (box_at_area ?box ?area) )
)
(:method (navigate ?to)
      ( (at ?from) )
      ( (navigate_ugv ?from ?to) )
)
(:method (navigate_ugv ?from ?to)
      ( (at ?to) )
      ( )
      ( (ground_path ?from ?to) )
      ( (!navigate ?from ?to) )
      ( ground_path ?from ?mid) (not (visited ?mid)) )
      ( (!navigate ?ugv ?from ?mid) (!!visit ?mid) (navigate_ugv ?ugv ?mid ?to) (!!unvisit ?mid) )
)
(:method (get_picture ?disaster)
      ( (disaster ?disaster) (visible_from ?disaster ?area) )
      ( (navigate ?area) (!take_picture ?area ?disaster) (send_data ?disaster) )
)
(:method (send_data ?disaster)
      ( (disaster ?disaster) (have_picture ?disaster) (cdm_at ?cdm ?area2) (area ?area2) (in_range ?
      area1 ?area2) )
      ( (navigate ?area1) (!communicate_data ?cdm ?disaster ?area1 ?area2) )
)
(:method (deliver_box ?box ?area)
      ( (box ?box) (area ?area) (store ?store) (box_at_cdm ?box ?cdm) (cdm_at ?cdm ?area2) )
      ( (navigate ?area2) (!pickup_box ?store ?cdm ?area2 ?box) (navigate ?area) (!drop_box ?store ?
      area ?box) )
)
))
```

Listing A.2 – HTN domain specification of an USV agent.

```
(defdomain floods-usv (
(:operator (!navigate ?from ?to)
      ( (area ?from) (area ?to) (water_path ?from ?to) (at ?from) )
      ( (at ?from) )
      ( (at ?to) )
)
(:operator (!take_picture ?area ?disaster)
      ( (area ?area) (disaster ?disaster) (visible_from ?disaster ?area) (at ?area) )
      ( )
      ( (have_picture ?disaster) )
)
(:operator (!communicate_data ?cdm ?disaster ?area1 ?area2)
      ( (disaster ?disaster) (cdm ?cdm) (at ?area1) (cdm_at ?cdm ?area2) (area ?area1) (area ?area2) (
      have_picture ?disaster) (in_range ?area1 ?area2) )
      ( (have_picture ?disaster) )
      ( (communicated_data ?disaster) )
)
(:operator (!sample_water ?store ?area)
      ( (store ?store) (empty ?store) (area ?area) (at ?area) )
      ( (empty ?store) )
      ( (full ?store) (have_water_sample ?area) )
)
(:operator (!drop_sample ?store ?area1 ?area2 ?cdm)
      ( (store ?store) (area ?area1) (area ?area2) (cdm ?cdm) (have_water_sample ?area2) (cdm_at ?
      cdm ?area1) (at ?area1) )
      ( (full ?store) (have_water_sample ?area2) )
      ( (empty ?store) (have_water_sample_cdm ?cdm ?area2) )
)
(:method (navigate ?to)
      ( (at ?from) )
      ( (navigate_usv ?from ?to) )
)
(:method (navigate_usv ?from ?to)
      ( (at ?to) )
      ( )
      ( (water_path ?from ?to) )
      ( (!navigate ?from ?to) )
      ( water_path ?from ?mid) (not (visited ?mid)) )
      ( (!navigate ?usv ?from ?mid) (!!visit ?mid) (navigate_usv ?usv ?mid ?to) (!!unvisit ?mid) )
)
(:method (get_picture ?disaster)
      ( (disaster ?disaster) (visible_from ?disaster ?area) )
      ( (navigate ?area) (!take_picture ?area ?disaster) (send_data ?disaster) )
)
(:method (send_data ?disaster)
      ( (disaster ?disaster) (have_picture ?disaster) (cdm_at ?cdm ?area2) (area ?area2) (in_range ?
      area1 ?area2) )
      ( (navigate ?area1) (!communicate_data ?cdm ?disaster ?area1 ?area2) )
)
(:method (deliver_sample ?cdm ?area2)
      ( (cdm ?cdm) (area ?area2) (store ?store) (cdm_at ?cdm ?area1) )
      ( (navigate ?area2) (!sample_water ?store ?area2) (navigate ?area1) (!drop_sample ?store ?
      area1 ?area2 ?cdm) )
)
))
```

Listing A.3 – HTN problem specification of an UGV agent.

```
(defproblem ugv1 floods-ugv (
    (area area1)
    (area area2)
    (area area3)
    (area area4)
    (disaster flood1)
    (disaster flood2)
    (disaster flood3)
    (disaster flood4)
    (cdm cdm1)
    (cdm_at cdm1 area1)
    (at area1)
    (ground_path area1 area3)
    (ground_path area3 area1)
    (visible_from flood1 area3)
    (visible_from flood2 area4)
    (visible_from flood3 area3)
    (visible_from flood4 area4)
    (in_range area2 area1)
    (in_range area1 area1)
)
(:unordered
    (:task get_picture flood3)
)
)
```

Listing A.4 – HTN problem specification of an USV agent.

```
(defproblem usv1 floods-usv (
    (area area1)
    (area area2)
    (area area3)
    (area area4)
    (disaster flood1)
    (disaster flood2)
    (disaster flood3)
    (disaster flood4)
    (cdm cdm1)
    (cdm_at cdm1 area1)
    (at area1)
    (water_path area1 area2)
    (water_path area2 area1)
    (water_path area2 area4)
    (water_path area4 area2)
    (visible_from flood1 area3)
    (visible_from flood2 area4)
    (visible_from flood3 area3)
    (visible_from flood4 area4)
    (in_range area2 area1)
    (in_range area1 area1)
)
(:unordered
    (:task get_picture flood4)
)
)
```

Listing A.5 – UGV agents' code snippet.

```
+!navigate(To)
    : at(From)
<-
    !navigate(From,To).

+!navigate(From,To)
    : at(To).

+!navigate(From,To)
    : ground_path(Areas)[artifact(From)] & .my_name(Name)
<-
    navigate(From,To,Areas)[artifact(Name)].

+!take_picture(Area,Disaster)
    : visible_from(Areas)[artifact(Disaster)] & .my_name(Name) & cdm_at(CdmAt)[artifact(Cdm)]
<-
    !navigate(Area);
    take_picture(Area,Disaster,Areas)[artifact(Name)];
    !communicate_data(Cdm,Disaster,Area,CdmAt).

+!communicate_data(Cdm,Disaster,At,To)
    : cdm_at(CdmAt)[artifact(Cdm)] & in_range(Areas)[artifact(Cdm)] & .my_name(Name)
<-
    !navigate(To);
    communicate_data(At,To,CdmAt,Disaster,Areas)[artifact(Name)].

+!deliver_box(Area,Box)
    : box_at(BoxAt)[artifact(Box)] & cdm_at(CdmAt)[artifact(BoxAt)] & .my_name(Name)
<-
    !navigate(CdmAt);
    pickup_box(CdmAt, CdmAt, BoxAt, BoxAt, Box)[artifact(Name)];
    !navigate(Area);
    drop_box(Area, Box)[artifact(Name)].
```

Listing A.6 – USV agents' code snippet.

```
+!navigate(To)
    : at(From)
<-
    !navigate(From,To).

+!navigate(From,To)
    : at(To).

+!navigate(From,To)
    : water_path(Areas)[artifact(From)] & .my_name(Name)
<-
    navigate(From,To,Areas)[artifact(Name)].

+!take_picture(Area,Disaster)
    : visible_from(Areas)[artifact(Disaster)] & .my_name(Name) & cdm_at(CdmAt)[artifact(Cdm)]
<-
    !navigate(Area);
    take_picture(Area,Disaster,Areas)[artifact(Name)];
    !communicate_data(Cdm,Disaster,Area,CdmAt).

+!communicate_data(Cdm,Disaster,At,To)
    : cdm_at(CdmAt)[artifact(Cdm)] & in_range(Areas)[artifact(Cdm)] & .my_name(Name)
<-
    !navigate(To);
    communicate_data(At,To,CdmAt,Disaster,Areas)[artifact(Name)].
+!deliver_sample(Cdm,Area2)
    : cdm_at(CdmAt)[artifact(Cdm)] & .my_name(Name) & store(Store)[artifact(Name)]
<-
    !navigate(Area2);
    sample_water(Area2)[artifact(Name)];
    !navigate(CdmAt);
    drop_sample(CdmAt, Area2, CdmAt)[artifact(Name)].
```

Listing A.7 – The UGV artefact.

```
1  public class VehicleUgv extends Artifact {
2      void init(String area, String store) {
3          defineObsProperty("at",area);
4          defineObsProperty("storage",store);
5      }
6      @OPERATION void navigate(String from, String to, String areas) throws
       InterruptedException {
7          ObsProperty cond1 = getObsProperty("at");
8          if (cond1(from) && areas.contains(to)) {
9              getObsProperty("at").updateValue(to);
10         } else { failed("Action navigate has failed."); }
11     }
12     @OPERATION void take_picture(String area, String disaster, String areas) throws
       InterruptedException {
13         ObsProperty cond1 = getObsProperty("at");
14         if (cond1(area) && areas.contains(area)) {
15             defineObsProperty("have_picture", disaster);
16         } else { failed("Action take_picture has failed."); }
17     }
18     @OPERATION void communicate_data(String at, String to, String cdmAt, String disaster,
       String areas) throws InterruptedException {
19         ObsProperty cond1 = getObsProperty("at");
20         ObsProperty cond2 = getObsPropertyByTemplate("have_picture", disaster);
21         if (cond1(at) && cond2(disaster) && to.equals(cdmAt) && areas.contains(to)) {
22             defineObsProperty("communicated_data", disaster);
23             removeObsPropertyByTemplate("have_picture", disaster);
24         } else { failed("Action communicate_data has failed."); }
25     }
26     @OPERATION void pickup_box(String cdmAt, String area, String boxAt, String cdm,
       ArtifactId boxId) throws InterruptedException, OperationException {
27         ObsProperty cond1 = getObsProperty("at");
28         ObsProperty cond2 = getObsProperty("storage");
29         if (cond1(area) && area.equals(cdmAt) && (boxAt.equals(area) || boxAt.equals(cdm)) &&
       cond2("empty")) {
30             getObsProperty("storage").updateValue("full");
31             execLinkedOp(boxId, "updateLoc", getCurrentOpAgentId().getAgentName());
32         } else { failed("Action pickup_box has failed."); }
33     }
34     @OPERATION void drop_box(String area, ArtifactId boxId) throws InterruptedException,
       OperationException {
35         ObsProperty cond1 = getObsProperty("at");
36         ObsProperty cond2 = getObsProperty("storage");
37         if (cond1(area) && cond2("full")) {
38             getObsProperty("storage").updateValue("empty");
39             execLinkedOp(boxId, "updateLoc", area);
40         } else { failed("Action drop_box has failed."); }
41     }
42 }
```

Listing A.8 – The USV artefact.

```
1  public class VehicleUsv extends Artifact {
2      void init(String area, String store) {
3          defineObsProperty("at",area);
4          defineObsProperty("storage",store);
5      }
6      @OPERATION void navigate(String from, String to, String areas) throws
       InterruptedException {
7          ObsProperty cond1 = getObsProperty("at");
8          if (cond1(from) && areas.contains(to)) {
9              getObsProperty("at").updateValue(to);
10         } else { failed("Action navigate has failed."); }
11     }
12     @OPERATION void take_picture(String area, String disaster, String areas) throws
       InterruptedException {
13         ObsProperty cond1 = getObsProperty("at");
14         if (cond1(area) && areas.contains(area)) {
15             defineObsProperty("have_picture", disaster);
16         } else { failed("Action take_picture has failed."); }
17     }
18     @OPERATION void communicate_data(String at, String to, String cdmAt, String disaster,
       String areas) throws InterruptedException {
19         ObsProperty cond1 = getObsProperty("at");
20         ObsProperty cond2 = getObsPropertyByTemplate("have_picture", disaster);
21         if (cond1(at) && cond2(disaster) && to.equals(cdmAt) && areas.contains(to)) {
22             defineObsProperty("communicated_data", disaster);
23             removeObsPropertyByTemplate("have_picture", disaster);
24         } else { failed("Action communicate_data has failed."); }
25     }
26     @OPERATION void sample_water(String area) throws InterruptedException {
27         ObsProperty cond1 = getObsProperty("at");
28         ObsProperty cond2 = getObsProperty("storage");
29         if ( (cond1(area)) && (cond2("empty")) ) {
30             getObsProperty("storage").updateValue("full");
31             defineObsProperty("have_water_sample", area);
32         } else { failed("Action sample_water has failed."); }
33     }
34     @OPERATION void drop_sample(String areaTo, String areaFrom, String cdmAt) throws
       InterruptedException {
35         ObsProperty cond1 = getObsProperty("at");
36         ObsProperty cond2 = getObsProperty("storage");
37         ObsProperty cond3 = getObsPropertyByTemplate("have_water_sample", areaFrom);
38         if (cond1(areaTo) && cond2("full") && cond3(areaFrom) && areaTo.equals(cdmAt)) {
39             getObsProperty("storage").updateValue("empty");
40             removeObsPropertyByTemplate("have_water_sample", areaFrom);
41             defineObsProperty("delivered_water_sample", areaFrom);
42         } else { failed("Action drop_sample has failed."); }
43     }
44 }
```

APPENDIX B – DOMAP PLANNER SCRIPTS

Listing B.1 shows the *startPlanner* Java class that uses the default Java runtime environment to start a new process that executes an Allegro script. This class is implemented as an internal action that is executed by Jason agents when they enter the individual planning phase of DOMAP. The error output is saved in a file for logging purposes. The solution is saved in a file that is passed along to the translator, who then parses it into plans to be added to the agent's plan library.

Listing B.1 – Java class to start the Allegro script.

```
1  public class startPlanner extends DefaultInternalAction {
2    public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
3      try
4      {
5        Runtime rt = Runtime.getRuntime();
6        Process proc = rt.exec("/home/acl/startPlanner.cl");
7        InputStream stderr = proc.getErrorStream();
8        InputStreamReader isr = new InputStreamReader(stderr);
9        BufferedReader br = new BufferedReader(isr);
10       String line = null;
11       while ( (line = br.readLine()) != null)
12         saveErrorFile;
13       InputStream stdin = proc.getInputStream();
14       InputStreamReader isr2 = new InputStreamReader(stdin);
15       BufferedReader br2 = new BufferedReader(isr2);
16       String line2 = null;
17       while ( (line2 = br2.readLine()) != null)
18         saveOutputFile;
19       int exitVal = proc.waitFor();
20       System.out.println("Process exitValue: " + exitVal);
21     } catch (Throwable t)
22       {
23       t.printStackTrace();
24       }
25     return true;
26   }
27 }
```

In Listing B.2 we show the Allegro script used to start SHOP2. The script is called from within the *startPlanner* Java class, and receives three variables: *$DOMAIN_NAME*, *$PROBLEM_NAME*, and *$TIMEOUT*. The *$DOMAIN_NAME* variable is obtained from the SHOP2 domain file that was provided by the translator. The *$PROBLEM_NAME* is retrieved from the SHOP2 problem file that was also provided by the translator. Finally, the *$TIMEOUT* variable is determined by the organisation.

Listing B.2 – Allegro CL script to start SHOP2.

```
#! /home/acl/alisp -#!
(require :asdf)
(push "/home/shop2/" asdf:*central-registry*)
(asdf:oos 'asdf:load-op :shop2)
(load "./$DOMAIN_NAME")
(define-$DOMAIN_NAME-domain)
(load "./$PROBLEM_NAME")
(find-plans '$PROBLEM_NAME :which :shallowest :time-limit $TIMEOUT)
```

APPENDIX C – ADDITIONAL RESULTS FOR THE ROVERS DOMAIN

Table C.1 – Plan size results collected from 20 runs per problem in the Rovers domain; best values are in bold font.

	ADP			**CMAP**	**DOMAP**			**PMR**	**SIW**
	Min	*Max*	*Avg*	*Size*	*Min*	*Max*	*Avg*	*Size*	*Size*
p01	**26**	**26**	**26**	**26**	28	35	32.15	32	31
p02	**42**	47	43.45	46	47	57	51.7	**43**	46
p03	**60**	64	62.2	**61**	79	101	90.8	67	66
p04	**74**	79	76.9	**75**	101	119	112.35	81	91
p05	**103**	107	106	**104**	112	140	127.35	115	118
p06	125	130	128.65	**119**	229	254	246.6	131	149
p07	148	156	151.9	**140**	217	262	236.65	153	155
p08	**142**	146	**144.3**	**145**	228	251	240.85	153	–
p09	**187**	201	**194.9**	**195**	288	328	312.55	200	231
p10	203	209	205.4	210	323	383	348.6	**189**	224

Table C.2 – Parallelism results collected from 20 runs per problem in the Rovers domain; best values are in bold font.

	ADP			**CMAP**	**DOMAP**			**PMR**	**SIW**
	Min	*Max*	*Avg*	*Para*	*Min*	*Max*	*Avg*	*Para*	*Para*
p01	22.333	22.333	22.333	**5.667**	**1.583**	12.917	**5.171**	128	27.583
p02	51.767	86.4	77.015	117.867	**20.4**	67.1	**35.5**	**44.967**	59.467
p03	97.143	111.143	106.98	**54.839**	**12.696**	105.839	**41.207**	232.839	86.214
p04	146.489	186.544	176.422	118.278	**2.233**	**62.489**	**24.019**	216.544	71.878
p05	246.629	248.265	247.65	**75.515**	**17.477**	105.659	**53.696**	206.629	131.061
p06	374.44	392.841	389.843	254.731	91.978	262.995	170.225	180.093	**72.401**
p07	347.2	379.45	365.456	100.867	**35.317**	95.329	57.466	180.663	**37.696**
p08	416.588	442.575	434.872	144.761	**11.438**	**103.556**	**42.009**	170.029	–
p09	322.576	416.421	374.244	263.461	**7.818**	61.042	**22.039**	184.947	**54.682**
p10	239.041	260.132	249.187	277.307	**73.108**	**265.11**	**127.278**	659.015	415.584

Table C.3 – Planning time results collected from 20 runs per problem in the Rovers domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	**0.242**	0.664	**0.276**	0.4	1.837	0.585	1.175	6.834	2.299	0.464	2.64	0.637	0.258	**0.418**	0.361
p02	**0.335**	**0.524**	**0.379**	0.519	0.706	0.661	1.029	2.948	1.401	1.125	1.262	1.209	0.389	0.64	0.539
p03	**0.47**	**0.704**	**0.506**	0.688	1.067	0.877	1.131	4.386	1.808	1.132	1.368	1.24	0.671	1.128	0.901
p04	**0.573**	**0.914**	**0.654**	0.955	1.247	1.089	1.335	6.333	2.282	1.66	2.137	1.753	1.153	1.964	1.638
p05	**1.214**	**1.286**	**1.234**	1.523	2.289	1.976	2.665	8.52	4.374	2.944	3.319	3.17	4.056	6.962	5.367
p06	2.675	**3.347**	2.772	**2.172**	3.395	**2.755**	3.256	18.806	7.896	5.464	7.785	6.05	9.664	16.494	12.93
p07	10.309	16.273	11.426	4.392	5.904	5.568	**2.046**	**4.085**	**2.976**	13.219	18.905	14.615	31.532	53.074	45.253
p08	5.35	8.616	6.14	3.291	4.936	4.257	**2.312**	**4.006**	**2.731**	8.312	11.619	9.077	–	–	–
p09	7.138	11.385	7.881	4.7	5.411	5.227	**2.637**	**4.788**	**3.406**	11.09	13.735	11.909	43	72.316	57.086
p10	14.39	23.072	16.305	5.962	9.276	7.707	**2.791**	**4.85**	**3.626**	22.969	34.649	26.914	124.569	208.285	190.267

Table C.4 – Execution time results collected from 20 runs per problem in the Rovers domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	6.656	6.812	6.709	**5.113**	**5.619**	**5.435**	6.116	9.166	6.636	12.132	12.875	12.572	7.619	9.145	8.177
p02	10.678	14.22	12.997	14.557	15.734	14.939	**7.596**	12.715	10.722	9.038	**9.639**	**9.153**	10.692	12.191	11.503
p03	12.867	14.791	13.688	11.688	**13.173**	**12.155**	**9.379**	20.71	15.426	21.608	23.211	22.435	18.308	20.652	19.242
p04	20.279	22.737	21.768	16.21	18.240	17.328	**11.336**	**14.882**	**13.112**	24.649	25.372	24.852	18.79	20.904	19.74
p05	28.286	30.363	29.203	16.058	**17.444**	16.611	**13.325**	18.328	**14.941**	20.21	22.336	21.103	27.482	30.445	29.053
p06	34.192	36.543	35.894	30.879	33.392	31.833	**18.789**	28.494	**22.160**	20.646	**25.344**	23.738	33.485	36.454	34.918
p07	29.022	36.187	31.549	17.998	**20.571**	**18.789**	**17.579**	24.383	20.406	22.031	26.758	24.875	29.955	33.495	31.43
p08	46.043	48.977	47.117	24.056	27.506	25.916	**18.848**	**25.914**	**20.653**	23.966	27.124	25.411	–	–	–
p09	38.636	44.647	41.267	36.089	43.077	39.567	**21.544**	**26.067**	**23.647**	27.425	32.968	29.519	44.734	49.842	46.638
p10	28.883	**33.858**	31.325	38.653	41.756	40.347	**22.641**	39.358	**27.595**	60.345	64.088	61.963	63.914	69.799	67.01

APPENDIX D – ADDITIONAL RESULTS FOR THE FLOODS DOMAIN

Table D.1 – Plan size results collected from 20 runs per problem in the Floods domain; best values are in bold font.

	ADP			**CMAP**	**DOMAP**			**PMR**	**SIW**
	Min	*Max*	*Avg*	*Size*	*Min*	*Max*	*Avg*	*Size*	*Size*
p01	43	48	46.3	45	43	60	53.6	**41**	45
p02	55	57	56.25	64	61	82	69.55	**53**	55
p03	117	122	118.2	119	**115**	142	129.35	**116**	130
p04	129	141	134.15	130	138	190	164.75	125	**124**
p05	168	171	168.7	**159**	182	239	203.05	160	168
p06	**176**	191	183.6	**179**	195	244	218.25	236	188
p07	**187**	205	196.25	**193**	215	247	234.7	201	233
p08	238	251	245.05	**221**	238	296	273.3	230	265
p09	302	319	311.3	**308**	**301**	352	329.2	346	326
p10	305	330	316.25	313	329	384	352.05	**294**	337

Table D.2 – Parallelism results collected from 20 runs per problem in the Floods domain; best values are in bold font.

	ADP			**CMAP**	**DOMAP**			**PMR**	**SIW**
	Min	*Max*	*Avg*	*Para*	*Min*	*Max*	*Avg*	*Para*	*Para*
p01	25.444	50.444	42.744	35.5	**2.75**	12.75	**7.919**	19.778	**11.5**
p02	38.242	42.265	38.841	99.697	**3.356**	15.97	7.357	32.811	**5.174**
p03	129.457	134.695	131.66	187.781	**20.6**	**42.41**	**32.519**	140.638	111.952
p04	120.918	133.712	127.959	181.242	**26.353**	50.408	37.405	146.644	**33.046**
p05	147.9	153.29	149.081	163.957	**31.857**	**58.314**	**43.966**	185.348	69.1
p06	189.536	259.259	220.326	204.346	**24.027**	**36.862**	**31.675**	223.362	80.928
p07	132.148	172.635	154.447	164.67	**15.422**	**26.84**	**20.524**	133.179	73.165
p08	235.72	297.126	272.175	202.516	**21.155**	**35.982**	**27.856**	188.644	45.868
p09	417.653	619.559	516.067	404.292	**31.127**	**48.28**	**40.811**	280.383	138.61
p10	261.52	514.729	336.33	223.131	**27.625**	**36.313**	**32.682**	200.623	68.313

Table D.3 – Planning time results collected from 20 runs per problem in the Floods domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	**0.411**	0.721	**0.469**	0.423	**0.534**	0.507	1.132	3.289	1.693	0.932	1.062	0.991	0.586	0.981	0.786
p02	0.786	1.3	0.922	**0.525**	**0.665**	**0.636**	1.3	3.076	1.661	1.197	1.393	1.267	1.57	2.677	2.124
p03	1.439	2.099	1.56	**0.743**	**1.088**	**0.918**	1.534	2.126	1.805	1.674	2.112	1.747	9.774	16.815	13.245
p04	2.779	4.237	3.118	**1.386**	**1.987**	**1.701**	1.759	2.932	2.101	2.64	3.232	2.783	21.096	35.566	27.927
p05	4.309	6.634	4.918	**1.606**	**2.238**	**1.895**	1.913	3.785	2.61	4.146	5.503	4.575	54.062	92.058	73.536
p06	6.06	9.994	7.086	2.387	**3.344**	**2.7845**	**2.218**	4.081	2.99	12.928	20.047	15.017	125.622	209.596	180.764
p07	9.511	12.612	10.835	**2.929**	**4.685**	3.726	2.948	5.573	**3.718**	7.414	9.398	7.811	285.255	479.705	429.912
p08	18.339	31.813	22.126	**3.534**	**4.812**	4.225	3.364	5.694	**4.175**	10.153	14.002	10.857	720.376	887.571	840.414
p09	26.783	41.332	33.815	5.067	7.789	6.119	**3.849**	**6.44**	**4.967**	55.333	81.05	67.477	1808.363	2016.963	1946.180
p10	40.279	136.793	51.971	7.512	11.243	9.296	**4.92**	**8.031**	**6.063**	17.761	25.928	19.334	2897.413	3304.565	3135.850

Table D.4 – Execution time results collected from 20 runs per problem in the Floods domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	4.061	4.315	4.201	4.049	4.272	4.142	4.077	4.715	4.286	**4.06**	4.343	4.168	4.064	**4.23**	**4.138**
p02	6.07	6.316	6.167	5.065	5.318	5.197	4.116	8.312	4.882	6.102	6.44	6.228	**4.091**	**4.36**	**4.188**
p03	2.183	2.618	2.346	2.823	3.08	2.937	3.158	5.908	4.285	1.831	2.125	1.97	**1.337**	**1.593**	**1.489**
p04	13.172	13.474	13.325	13.214	13.422	13.335	**7.227**	11.428	**9.5**	14.207	14.488	14.321	10.107	**10.506**	10.298
p05	4.209	5.575	5.22	**4.16**	**4.597**	**4.346**	6.258	9.38	7.049	5.211	5.406	5.29	4.736	5.021	4.822
p06	29.328	31.414	29.649	29.356	29.776	29.470	**8.633**	**12.466**	**11.06**	20.281	20.602	20.455	23.338	23.679	23.441
p07	17.332	22.485	19.962	22.353	22.61	22.478	**6.912**	**9.178**	**8.482**	16.41	16.693	16.5	19.906	20.057	19.974
p08	28.46	28.752	28.593	25.495	25.711	25.589	**8.579**	**12.642**	**10.983**	23.482	23.835	23.629	16.484	16.777	16.579
p09	**5.616**	**6.838**	**6.556**	10.831	11.347	11.157	8.618	10.796	9.759	11.553	11.736	11.646	13.545	13.752	13.642
p10	28.709	55.24	32.736	19.075	19.453	19.287	**10.681**	**13.266**	**11.51**	22.749	22.987	22.802	20.621	21.033	20.83

APPENDIX E – ADDITIONAL RESULTS FOR THE PETROBRAS DOMAIN

Table E.1 – Plan size results collected from 20 runs per problem in the Petrobras domain; best values are in bold font.

	ADP	**CMAP**	**DOMAP**			**PMR**	**SIW**
	Size	Size	Min	Max	Avg	Size	Size
p01	**16**	–	**15**	**16**	**15.25**	–	18
p02	**19**	**19**	**19**	20	19.05	**19**	21
p03	**22**	**22**	**22**	24	22.65	**22**	26
p04	**25**	**25**	**25**	27	26	**25**	31
p05	**27**	**27**	28	30	29.15	**27**	34
p06	**30**	**30**	33	35	33.45	**30**	37
p07	**33**	**33**	35	39	37.1	**33**	42
p08	**35**	**35**	38	43	41.3	**35**	48
p09	**37**	**37**	42	47	44.6	**37**	51
p10	42	**40**	47	52	48.45	**40**	55

Table E.2 – Parallelism results collected from 20 runs per problem in the Petrobras domain; best values are in bold font.

	ADP	**CMAP**	**DOMAP**			**PMR**	**SIW**
	Size	Size	Min	Max	Avg	Size	Size
p01	64	**–**	**0**	18.917	13.121	–	10.667
p02	72.2	72.2	**6.2**	14.2	12.69	72.2	**8**
p03	80.667	80.667	**0**	16.267	11.795	80.667	6.567
p04	89.286	89.286	**8.238**	13.571	13.035	89.286	**12.238**
p05	81	81	11.361	15.361	14.116	81	**8.111**
p06	90	90	**9.877**	14.044	13.149	90	**10.667**
p07	99	99	8.891	15.872	13.146	99	**6.491**
p08	102.083	102.083	**8.969**	15.295	**12.403**	102.083	**12.629**
p09	105.308	105.308	**8.231**	15	**11.733**	105.308	**14.141**
p10	126	114.286	**7.604**	**14.901**	**12.481**	114.286	15.209

Table E.3 – Planning time results collected from 20 runs per problem in the Petrobras domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	0.323	**0.462**	0.361	–	–	–	1.032	2.126	1.375	–	–	–	**0.229**	0.566	**0.315**
p02	0.398	**0.461**	0.415	**0.304**	0.664	**0.357**	1.146	1.474	1.394	0.361	0.952	0.449	0.315	0.527	0.407
p03	0.491	0.708	0.531	**0.324**	**0.384**	**0.366**	1.255	1.548	1.393	0.393	0.462	0.448	0.449	0.746	0.666
p04	0.605	0.882	0.646	**0.342**	**0.495**	**0.403**	1.037	1.516	1.408	0.473	0.589	0.499	0.62	1.036	0.89
p05	0.723	1.092	0.796	**0.375**	**0.523**	**0.415**	1.14	1.573	1.414	0.468	0.545	0.528	0.828	1.403	1.194
p06	0.863	1.261	0.968	**0.403**	**0.565**	**0.459**	1.43	1.583	1.492	0.548	0.602	0.577	1.091	1.858	1.521
p07	1.005	1.501	1.073	**0.413**	**0.5**	**0.473**	1.279	1.611	1.494	0.547	0.643	0.612	1.514	2.618	2.285
p08	1.189	1.874	1.354	**0.441**	**0.638**	**0.499**	1.485	1.631	1.579	0.605	0.697	0.672	2.093	3.548	3.099
p09	1.341	2.026	1.472	**0.49**	**0.703**	**0.569**	1.258	1.827	1.616	0.673	0.947	0.745	2.718	4.612	3.768
p10	1.528	2.232	1.639	**0.503**	**0.591**	**0.561**	1.322	1.731	1.623	0.708	1.04	0.827	3.587	6.162	5.081

Table E.4 – Execution time results collected from 20 runs per problem in the Petrobras domain; best times are in bold font.

	ADP			**CMAP**			**DOMAP**			**PMR**			**SIW**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
p01	8.047	8.169	8.088	–	–	–	**2.1**	**4.109**	**3.819**	–	–	–	4.021	4.149	4.066
p02	9.557	9.672	9.61	9.539	9.774	9.625	**3.549**	**4.105**	**3.986**	9.538	9.871	9.608	4.021	4.195	4.079
p03	11.048	11.337	11.148	11.055	11.239	11.135	**2.202**	4.235	**3.944**	11.043	11.374	11.151	4.026	**4.204**	4.104
p04	12.553	12.751	12.639	12.555	12.801	12.632	**3.613**	**4.257**	**4.064**	12.572	12.786	12.657	4.525	4.71	4.615
p05	13.597	13.906	13.726	13.568	13.836	13.664	**4.041**	**4.255**	**4.134**	13.597	13.861	13.707	4.111	4.355	4.181
p06	15.074	15.312	15.155	15.057	15.325	15.169	**4.023**	**4.257**	**4.159**	15.063	15.347	15.163	4.039	4.306	4.209
p07	16.598	16.818	16.723	16.595	16.895	16.705	**4.065**	4.325	**4.183**	16.593	16.808	16.685	4.097	**4.313**	4.184
p08	17.6	17.864	17.716	17.58	17.805	17.678	**3.745**	**4.273**	**4.149**	17.604	17.817	17.716	5.534	5.782	5.63
p09	18.606	18.871	18.732	18.584	18.895	18.736	**3.727**	**4.34**	**4.163**	18.628	18.855	18.713	5.573	5.839	5.679
p10	21.137	21.393	21.255	20.122	20.361	20.221	**3.75**	**4.307**	**4.179**	20.1	20.339	20.211	5.585	5.841	5.673

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br