A Distributed Online Multi-Agent Planning System

Rafael C. Cardoso and **Rafael H. Bordini**
School of Informatics – FACIN-PPGCC
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil
{rafael.caue@acad.pucrs.br, rafael.bordini@pucrs.br}

Abstract

Multi-agent planning is an important capability to have in the development of multi-agent systems, which still remains an open problem, mainly because of the gap between planning and execution. Multi-agent systems often have dynamic environments that require planning to be done during run-time (i.e., online planning). In this paper, we use a platform for the development of multi-agent systems (JaCaMo) in both decentralised multi-agent planning and execution stages, providing a multi-agent system with capabilities to solve online multi-agent planning problems. The contributions shown in this paper are: i) the design of a Distributed Online Multi-Agent Planning System (DOMAPS); ii) the implementation of DOMAPS in JaCaMo; and iii) initial experiments in the Floods domain, a novel planning domain that uses heterogeneous unmanned vehicles to respond to flood disasters.

1 Introduction

Multi-Agent Systems (MAS) are often situated in dynamic environments where new plans of actions need to be constantly devised in order to successfully achieve the system's goals. Therefore, employing planning techniques during run-time of a MAS can be used to improve agent's plans using knowledge that was not previously available, or even to create new plans to achieve some goal for which there was no known course of action at design time.

Research on automated planning has been mostly focused on single-agent planning over the years. Although it is possible to adapt centralised single-agent techniques to work in a decentralised way, such as in (Crosby, Jonsson, and Rovatsos 2014), distributed computation is not the only advantage of using Multi-Agent Planning (MAP).

In MAP, by allowing agents to do their own individual planning (i.e., planning by multiple agents), the search space can be effectively pruned, which can potentially decrease planning time on domains that are intrinsically distributed. This 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. The output of a MAP process are plans for multiple agents. Single-agent planning for multiple agents can have no privacy, since the planner needs all the information available, and the agents in the problem representation are usually considered as any other object of the environment. These differences are characterised in Table 1, based on the descriptions found in (Durfee and Zilberstein 2013).

MAS went through a similar process of transitioning from single to multiple agents, albeit at a faster rate. Recent research, as evidenced in (Boissier et al. 2011; Singh and Chopra 2010), shows that considering other programming dimensions such as environments and organisations as first-class entities along with agents allow developers to create more complex MAS.

In this paper, we introduce the design of our Distributed Online Multi-Agent Planning System (DOMAPS). DOMAPS is composed of: i) a formalism for the representation of domains and problems in online multi-agent planning, based on Hierarchical Task Network (HTN); ii) a contract net protocol mechanism for goal allocation; iii) individual planning with the SHOP2 planner; and iv) the use of social laws to coordinate the agents during execution. Some preliminary results from experiments in a novel scenario, the Floods domain, are shown.

Although approaches to online single-agent planning usually involve some kind of interleaving planning and execution, we focus on domains that allow agents some time to plan while the system is still in execution (i.e., anytime planning). DOMAPS allows for the dynamic execution of plans found during run-time, making it easy to transition from planning into execution and vice-versa, while still permitting agents to continue their execution, as long as their actions are believed not to cause any conflict with actions from a possible solution.

In DOMAPS current configuration, during the planning stage the environment is assumed to be deterministic and fully observable (although each agent has its own perspective of the environment). However, in the execution stage we do not make these assumptions, the environment can be non-deterministic and actions can fail, in which case it may be necessary to replan.

The remainder of the paper is structured as follows. In the next section a brief description of the MAS development platform, JaCaMo, used to implement DOMAPS and run the agents and the MAS is given. Section 3 introduces the initial design of the Distributed Online Multi-Agent Planning System. Next, in Section 4, we describe the implementation of

Table 1: 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 entities

DOMAPS in JaCaMo. In Section 5, we describe the Floods domain, and some initial experiments of using DOMAPS in this domain. We follow with a discussion on related and future work, and end the paper with some concluding remarks.

2 Background

DOMAPS was designed for online systems, thus, requiring the use of planning techniques whilst the MAS is running. Therefore, we need a MAS development platform in order to properly implement and evaluate DOMAPS. We chose to use the **JaCaMo**[1] (Boissier et al. 2011) MAS development platform, since it contains programming abstractions that we found to be a suitable match for the implementation of DOMAPS – **organisation**, **environment**, and **agent** abstractions.

JaCaMo combines three separate technologies into a platform for MAS programming that makes use of multiple levels of abstractions, enabling the development of robust MAS. Each technology (Jason, CArtAgO, and Moise) was developed separately for a number of years and are fairly established on their own when dealing with their respective abstraction level (agent, environment, and organisation).

Moise (Hübner, Sichman, and Boissier 2007) handles the organisation level, and how to specify an organisation in a MAS. This level adds first-class elements to the MAS such as roles, groups, organisational goals, missions, and norms. Agents can adopt roles in the organisation, forming groups and sub-groups. Missions are defined to achieve the organisation goals. The behaviour of the agents that adopt roles to execute these missions is guided by norms.

Jason (Bordini, Wooldridge, and Hübner 2007) is responsible for the agent level. It is an extension of the AgentSpeak language, based on the BDI 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.

CArtAgO (Ricci et al. 2009) is based on the A&A (Agents and Artefacts) model (Omicini, Ricci, and Viroli 2008), and deals with the environment level. Artefacts are used to represent the environment, storing information about the environment as observable properties and providing actions that can be executed through operations. Agents can focus on specific artefacts in order to obtain information contained on that artefact. When an agent focuses on an artefact, it receives the observable properties as beliefs, and it is able to execute the artefact's operations.

[1]`http://jacamo.sourceforge.net/`.

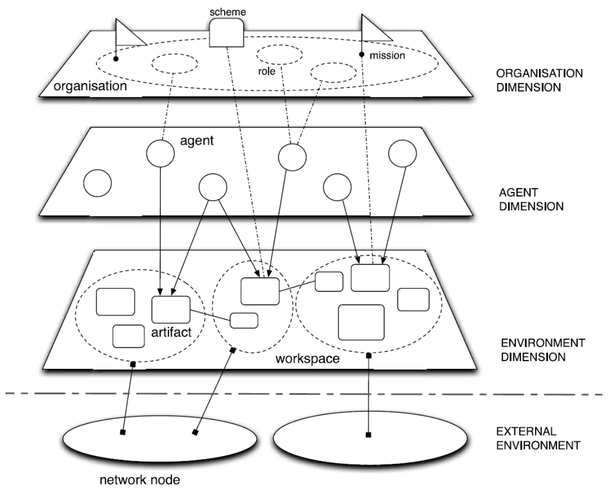

Figure 1: The JaCaMo overview (Boissier et al. 2011).

An overview of how JaCaMo combines these different levels of abstraction can be observed in Figure 1. 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 the agents that inhabit the agent dimension. In the bottom-most dimension, the environment houses artefacts that represent objects and information about the environment, grouped by workspaces that agents can access. These workspaces can be distributed across multiple network nodes, providing the distribution of the MAS.

3 The Distributed Online Multi-Agent Planning System

Our framework, the Distributed Online Multi-Agent Planning System (DOMAPS), consists of four main components: **planning formalism** – a formal representation of the information from the planning domain and problem that will be used during planning; **goal allocation** – the mechanism used to allocate goals to agents; **individual planning** – the planner used during each agent's individual planning stage; and **coordination mechanism** – used before or after planning to avoid possible conflicts that can be generated during planning.

DOMAPS was made to work as a general-purpose domain-independent system, and as such, we expect to turn it into an open platform where many other alternatives for main components can be added, allowing MAS developers and researchers to pick and choose the ones that work better to solve their online multi-agent planning problem.

The design overview of DOMAPS is shown in Figure 2. 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 receive role-related missions, while pursuing the organisation's goals. These aspects are what represent the MAS part of the application.

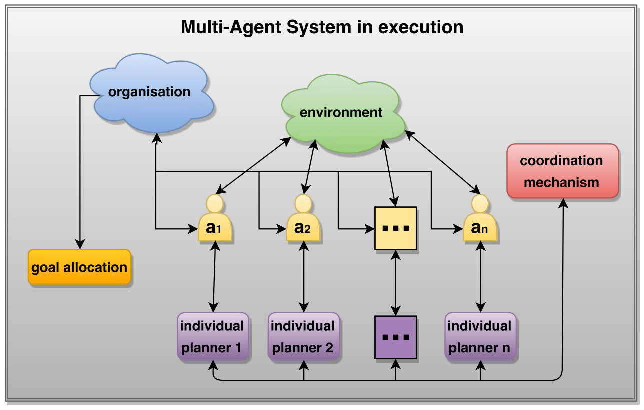

Figure 2: DOMAPS design overview.

Planning input (i.e., domain and problem representation) and output (i.e., the solution) are regulated by a planning formalism. The agents themselves plan individually, using an appropriate planner for the planning problem at hand. Coordination is used in order to achieve the organisational goals of the system. For example, goals that depend on joint plans involving multiple agents, or an agent's actions that can cause conflict with the other agents' plans. Since we are dealing with planning during run-time (online), new organisational goals can emerge (or their conditions can change) during the execution of the MAS. Therefore, we also use a mechanism to allocate these organisational goals to the appropriate agents, that is, allocate them to the agents that have an estimated better chance at solving that particular goal.

During execution, the creation of new organisational goals can start the planning process in DOMAPS, which consists of the following steps:

1. **Allocate goals:** Agents gain access to the organisational goals that will be planned for. Then, a mechanism is used to separate and allocate goals to agents that have the most (estimated) chance of finding a potential solution to the goal.

2. **Obtain up-to-date information needed for planning:** Environment and world information need to be collected from the MAS in execution, and translated into a planning formalism that can be used by the planner. Since planners work individually, each agent passes its input to their respective planner. Thus, the information that a planner has access to is limited to the information that the agent is allowed to access during that exact moment in the execution. Consequently, this ensures, at least, some partial privacy in DOMAPS.

3. **Agents start their individual planner:** The planner starts its search for a solution to the allocated goal, or set of goals.

4. **Coordination before or after planning:** Agents coordinate with each other before or after the planning process, in order to prevent any conflicts or help solve any dependencies.

5. **Translate solution:** Each agent translates the solution found by their respective planners into plans that can be added to their plan library. If we are dealing with coordination after planning, then any points of conflict or dependency found along the way should be sent to the coordination mechanism. Otherwise, when coordination is done before planning, the solution should already be free of conflicts and dependencies.

Another goal that we have with DOMAPS, is to allow the addition of new approaches to each of the four main components more easily, that way it is possible to choose the approach that is more suited for a particular problem. Next, we describe the initial approach used for each component, and discuss alternative approaches in Section 6.

3.1 Planning Formalism

We developed the Multi-Agent Hierarchical Task Network (MA-HTN) formalism, which is an extension of the single-agent HTN formalism used in the SHOP2 planner (Nau et al. 2003). MA-HTN is intended for **online** multi-agent planning problems, since domain and problem information have to be collected during execution. Agents use a **translator** to parse their information about the world into domain and problem specifications that is then passed to their own individual planner. The MA-HTN grammar for the problem and domain representation, as well as the translator's specification, can be found in (Cardoso and Bordini 2016).

Each agent has their own problem and domain specification. This provides a decent level of privacy on its own, since each planner only has access to their respective agent problem and domain specifications. This means that, unlike some of the other multi-agent planning formalisms, MA-HTN does not need to have private or public blocks. Although at some point it might be interesting to add the capability to include private goals, for now we are interested only on searching solutions for organisational goals.

Actions from other agents can cause conflicts, either at the moment that the action is executed (e.g., concurrent actions) or in the future (e.g., durative actions). Actions that can cause **conflict** have to be annotated by the MAS developer, in order for the translator to identify them. Likewise, dependencies between actions can also exist, either as a concurrent action that requires another agent or as actions that depend on the actions of other agents to happen first. These **dependency** relations also have to be annotated by the MAS developer, so that the translator can add them to the specification. Both conflicts and dependencies specifications are used by the coordination mechanism to coordinate the agents.

3.2 Goal Allocation

A Contract Net Protocol (CNP) mechanism is used to allocate goals to agents in DOMAPS. Our CNP mechanism is based on the original CNP design of Reid G. Smith (Smith 1980), with a few modifications in order to accommodate our needs for a goal allocation mechanism in the context of MAP. The initiator in our case is the organisation. It is the organisation's role to start new auctions for organisational goals that do not have any known plans on how to achieve the goals, or for organisational goals that have plans, but need to be re-planned. The bidders are agents from the organisation that also participate in the planning process.

The logic for determining an agent's bid depends on the rest of the mechanisms being used in DOMAPS and in the MAS development platform, but it is fair to assume that agents have the ability of checking their plan library for plans that are able to decompose, at least at some level, the goal that is being auctioned. Although domain-dependent functions for determining the bid can, generally, provide better results, we supply a simple domain-independent general-purpose function that agents can use to determine their bid, shown in Algorithm 1.

The agent checks if the announcement of the goal came from the organisation and if it is eligible, according to the eligibility criteria provided in the announcement, or otherwise decides not to bid. If the agent chooses to proceed with the bid, then, he keeps decomposing the goal into subtasks and incrementing the bid by 1 for each level that was successfully decomposed, either until it is close to the deadline, or it arrived in an action that could achieve the goal, or it found a dead end (in which case the bet becomes empty). The recursion indicates the backtrack when there are no more levels.

The initiator allocates the goal to the agent with the lowest (not empty) bid. We found that the lowest bid heuristic had better results in our initial experiments, as opposed to using the highest bid heuristic. Our logic behind using the lowest, is that it indicates that lower bids from agents, with different plan libraries and who were able to arrive at final decompositions, means that they can arrive at the solution using the lowest number of actions. However, for homogeneous agents with similar plan libraries and who were not able to fully explore their plan library, the selection of the highest bid may yield better results, since it would represent the agent who could decompose the furthest. We assume here that every goal can eventually be allocated, meaning that there is at least one agent eligible (and capable) for each organisational goal.

Regarding plan decomposition, agents do not check the plan's context (preconditions), nor does it uses any action theory to simulate future states. It simply decomposes into the first plan found in the plan library, and then, proceeds to decompose into any first plan found in the body of the original plan. Once there are no more levels to decompose, the agent backtracks to the original plan and chooses another branch, if there is one.

Algorithm 1 Domain-independent algorithm for determining an agent's bid.

```
function bid (bid-value, from, goal, eligibility, deadline)
if ((from $\neq$ organisation) or (not eligible)) then
    return bid-value $\leftarrow \emptyset$
else
    while ((close to deadline) or (no more levels available to decompose)) do
        decompose one level of one task from goal
        bid-value $\leftarrow$ bid-value + 1
        if deadend then
            return bid-value $\leftarrow \emptyset$
        end if
    end while
    if close to deadline then
        return bid-value
    end if
    if there are more levels available to decompose then
        bid (bid-value, from, goal, eligibility, deadline)
    end if
    return bid-value
end if
```

3.3 Individual Planner

SHOP2 (Nau et al. 2003) is a HTN planner with support for anytime planning. No modifications were made to the actual planning algorithm and search heuristics of SHOP2. Instead, we use the mechanisms from the other components to handle the multi-agent part of the planning process. By making little to no modifications to the individual planners, DOMAPS benefits from its multi-layered approach, making it easier to swap components without having to modify the planner's code directly.

Many parameters can be used to tweak the SHOP2 planner. The most relevant to DOMAPS is the parameter that guides which kind of search will be made:

- **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.

3.4 Coordination Mechanism

Social laws can coordinate agents by placing restrictions on the activities of the agents within the system. The purpose of these restrictions are twofold: it can be used to prevent some destructive interaction from taking place; or it can be used to facilitate some constructive interaction.

The design of social laws is domain-dependent, and we require them to be supplied by the system designer offline (i.e., they are provided before planning). We apply the social laws for the coordination of agents after planning, during execution.

In the original model of Shoham and Tennenholtz (Shoham and Tennenholtz 1995), 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 organisational goals, and thus, are naturally compelled to follow the social laws that are present in the system.

We formally define social laws in our model as:

Definition 1 Given a set of agents $\mathtt{Ag}$, a set of actions $\mathtt{Ac}$, a set of states $\mathtt{S}$, a set of preconditions $\mathtt{P}$, and a set of options Θ, a *social law* is a tuple ($\mathtt{ag}$,$\mathtt{ac}$,$\mathtt{s}$,$\mathtt{P}$,Θ) where $\mathtt{ag} \in \mathtt{Ag}$, $\mathtt{ac} \in \mathtt{Ac}$, and $\mathtt{s} \in \mathtt{S}$.

A social law $\mathtt{sl}$ constrains a specific action $\mathtt{ac}$ of agent $\mathtt{ag}$, considered to be a possible point of conflict (as established in the operator description from the MA-HTN formalism). When the state $\mathtt{s}$ satisfies each precondition $\rho_i \in \mathtt{P}$, the agent is given all possible options $\theta_i \in \Theta$. Although not explicitly present in this model, the *null* action (i.e., do nothing) can be a possible option, but in order for it to be viable it needs to have been established as an action in the MAS.

4 Multi-Agent System Integration

We implemented the `domaps.plan` internal action to start the DOMAPS planning process – internal actions are actions that Jason agents can execute internally, as opposed to external actions, which are environment-related. These internal actions are implemented as Java classes that agents can call.

To illustrate the run-time of DOMAPS when the `domaps.plan` internal action is executed, consider the overview provided in Figure 3. When an agent calls `domaps.plan`, it goes through phase 1 and activates the contract net protocol mechanism to allocate organisational goals between the agents. Then, in phase 2, each agent knowledge about the world is passed to a MA-HTN translator, that sends the information needed to SHOP2 for the individual planning that takes place in phase 3. The solution found by each agent's planner goes back through the MA-HTN translator again, translating the solution into AgentSpeak Jason plans. Finally, the solution is carried out by the agents, in accordance to the social laws (phase 5) that are associated with the actions that can cause conflicts.

4.1 MA-HTN

Each agent uses a MA-HTN translator in order to parse the current information obtained from the CArtAgO environment artefacts, as well as from their own personal artefact, and the plans from its plan library. The MA-HTN translator generates a problem and domain representation for each agent, as follows:

- **Problem representation:** The name of the problem and the name of the domain are obtained dynamically. The name of the agent is the one who started the translator.

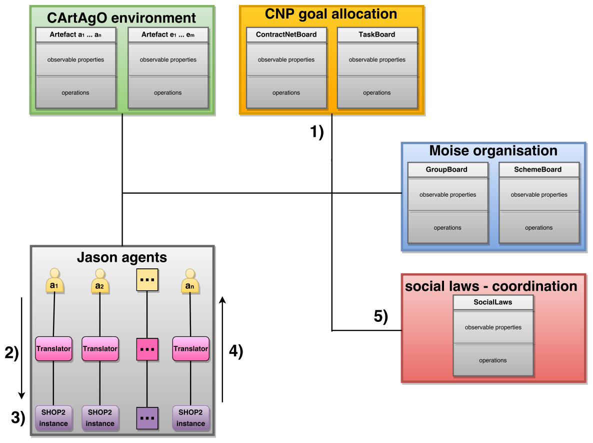

Figure 3: DOMAPS run-time overview of the `domaps.plan` internal action.

  The information collected from the CArtAgO artefacts are parsed into the agent's facts and initial states. The goal list is created from the organisational goals that were assigned to this particular agent during the goal allocation phase.

- **Domain representation:** The name of the domain is obtained dynamically. The name of the agent is the one who started the translator. Operators are parsed from all of the artefacts operations that the agent has access to. The preconditions are obtained from any conditional tests in an operation, the delete and add list are acquired from the deletion and addition of observable properties, respectively. The conflict and dependency lists need to be previously annotated into the operation in order for them to be able to be parsed. The methods are parsed from all of the plans in the agent's plan library, with the preconditions parsed from the context of the plan, and the task list parsed from the body of the plan.

4.2 Contract Net Protocol

The contract net protocol artefacts mediate the goal allocation phase of DOMAPS. The mechanism is represented by two artefacts: the *TaskBoard* artefact and the *ContractNetBoard* artefact. All of the agents that will participate in the planning stage take the roles of *bidders*. The *bidders* should always focus on the *TaskBoard*, as that is the artefact in which the organisational goals are announced. The role of *initiator* is restricted to the organisation. When the *initiator* announces an auction for a new organisational goal, it creates a *ContractNetBoard* associated with that goal.

We show the observable properties and operations of the *TaskBoard* and the *ContractNetBoard* artefacts in Figure 4. When a new new task is announced by the *initiator*, a *task* observable property is created. 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 organisation artefact, as the *initiator*, can announce goals in the *TaskBoard*. When the auction process ends, the *initiator* performs the *clear* operation to delete the observable property associated with that goal. This also generates an event in Jason, a belief deletion event, in which agents can perform clean-up and some other necessary activities.

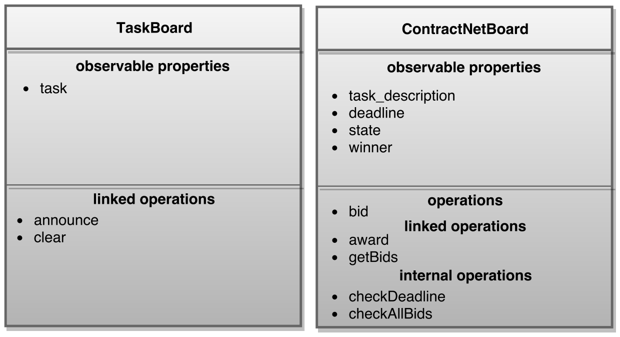

Figure 4: The task board and CNP board artefacts.

A *ContractNetBoard* is created for each goal announced by the *initiator*. The *bidder* agents focus on these new artefacts as soon as they perceive that a new goal was announced. The *task_description* contains an organisational goal, *deadline* is the time (in milliseconds) that the auction will run for, *state* informs if the auction is still open or not, and *winner* is created by the *initiator* once the auction ends with the id of the bid that won the auction.

The operation *bid* is executed by *bidders* in order to place a bid for the goal associated with the artefact. *award* is a linked operation executed by the *initiator*. It updates the *winner* observable property, based on a value function. The *getBids* is a linked operation executed by the *initiator*, returning all bids currently placed by the *bidders*, to be used in the *award* operation.

There are also two internal operations, *checkDeadline* and *checkAllBids*. Both internal operations update the *state* observable property to closed, if the deadline is up or if all agents have already placed a bid. Internal operations are not available to be used by agents or other artefacts. Instead, the artefact's operations themselves can trigger the asynchronous execution of internal operations.

4.3 SHOP2

We did not modify directly any of the SHOP2 code. We provide a *startPlanner* Java class that uses the default Java runtime environment to start a new process that executes an Allegro CL script in order to run SHOP2. The Java class is implemented as an internal action that is executed by Jason agents when they enter their individual planning stage.

4.4 Social Laws

In Figure 5, we show the observable properties and operations of the *SocialLaws* artefact. This artifact is responsible for coordinating the agents during the execution of a solution found during the planning process. It is created during the system's initialisation, one instance for each social law.

The observable properties are: *social_law* contains the name of the social law; *action_name* is the name of the action that is associated with this social law artefact; *precondition_list* is the list of preconditions that make this social law applicable; and *action_options* contains the list of possible actions that an agent may take in order to avoid a conflict, or to solve a dependency.

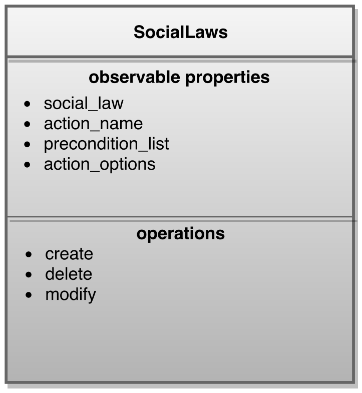

Figure 5: The artefact for social laws.

We also provide operations related to the manipulation of social laws, although there is no mechanism implemented to make use of these operations yet. 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 used.

Regarding the practical usage of the artefact, agents consult the *SocialLaws* artefact associated with the action that they are about to execute. This process is only necessary for actions that are part of the plans adopted as a solution from the planning stage, and only if those actions are annotated with conflict and/or dependency flags.

5 The Floods Domain

The lack of complex multi-agent domains led us to design a new domain, in order to best exploit the advantages of MAP and MAS. The inspiration for this specific domain came from a real-world scenario on using artificial intelligence techniques (e.g., a team of autonomous multi-robots) to help mitigate and prevent natural disasters. This scenario is specifically targeted at flood disasters, often caused by intense hydro-meteorological hazards, that can lead to severe economic losses and in some extreme cases even deaths.

Our domain, the Floods domain, is based on that real-world scenario. Another source of inspiration was the Rover domain, which was used in several past International Planning Competitions (IPCs). In the floods domain, a team of autonomous and heterogeneous robots are dispatched to monitor flood activity in a region. The Centre for Disaster Management (CDM) establishes a base of operation in the region that is being monitored. The base is used to assign goals to the robots, receive and interpret data, and provide some assistance. The CDM is usually operated by humans, but in our JaCaMo+DOMAPS implementation we simulate them by using agents, capable of creating dynamic goals during run-time.

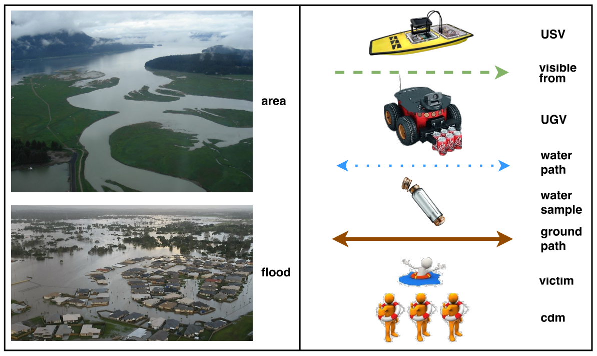

Figure 6: Elements from the Floods domain.

In Figure 6, we show the elements that compose the Floods domain. The domain takes place in a particular region, which is divided into several interconnected *areas*. Movement through the region occurs from traversing these areas. *Flood* events are common in the region, especially during heavy-rain. These floods can be observed from specific areas in the region. The areas can be connected by a *water path*, that can be traversed by naval units, and/or by a *ground path*, that can be traversed by ground units. *Water sample* can be requested to be collected from certain areas. During flood events, *victims* may be detected and in need of assistance. The *CDM* establishes a base of operations in one of the areas in the region.

Finally, the naval units are composed of Unmanned Surface Vehicles (*USVs*) that can move through areas connected by water paths, collect water samples, and take pictures of flood events. Meanwhile, the Unmanned Ground Vehicles (*UGVs*) are ground units that are able to move through areas connected by ground paths, take pictures of flood events, and provide assistance to victims by transporting first-aid kits to first responders close by. The robots can only perceive other robots that are in the same area.

5.1 Initial Experiments

For the initial experiments presented here, we maintained the number of agents and focused on increasing the number of goals. It seems that there is a relation between the number of goals and the number of agents. For most domains, having the number of goals equal to the number of agents, and assuming that each agent is capable of solving its associated goal, appears to result in faster planning times. In Table 2 we show the some initial experiments on this domain for small problems with 4, 8, 16, and 32 goals. As the number of goals surpasses the number of agents, the planning time approximates to that of single-agent SHOP2.

The results are shown in regards to time spent planning, and the number of state expansions and inferences that were made during planning. These results do not depict any of the run-time features of DOMAPS, as we are still investigating how to evaluate it as a whole, and considering what evaluation parameters that could be used both for planning and execution in tandem.

Table 2: Initial experiment results.

	DOMAPS			**SHOP2**
	usv1	usv2	ugv1	
floods 4 pl. time	0.001	0.001	0.001	0.004
exp.	8	8	15	65
inf.	13	13	21	186
floods 8 pl. time	0.001	0.001	0.002	0.011
exp.	15	15	29	129
inf.	21	21	37	360
floods 16 pl. time	0.002	0.002	0.004	0.033
exp.	29	29	57	257
inf.	37	37	69	708
floods 32 pl. time	0.003	0.003	0.005	0.095
exp.	57	57	113	513
inf.	69	69	133	1404

It is natural for DOMAPS to have faster planning times than regular SHOP2 since we assign goals to agents before planning, while SHOP2 does so during planning, in order to try different assignments. In future experiments we want to test scalability and add more domains. We also hope to compare DOMAPS with other frameworks, and provide a full framework evaluation, as well as evaluating each of its components separately.

These experiments show an interesting result in regards to the number of expansions and inferences. DOMAPS requires fewer state expansions and inferences, even if adding all the agents, than SHOP2. The individual planning approach taken in DOMAPS discards many of the predicates that are usually used to assign tasks between different objects, whilst SHOP2 needs those predicates to define the (agent) objects. By considering agents as first-class abstractions in MA-HTN, we are free of the use of these predicates.

6 Related and Future Work

There has been several surveys over the years describing advancements in particular areas of planning. Of interest, and related to this research, there are: in (desJardins et al. 1999), a survey on distributed online (continual) planning is presented, with the state of the art in distributed and online planning at the time, and a conceptual design for distributed online planning; a survey (Meneguzzi and De Silva 2013) that presents a collection of recent techniques used to integrate single-agent planning for a single-agent in BDI-based agent-oriented programming languages, focusing mostly on efforts to generate new plans at run-time; and a multi-agent planning survey (Weerdt and Clement 2009), describing several approaches taken towards multi-agent planning over the last few years.

In (Nissim and Brafman 2014), the authors propose a forward search heuristic for classical multi-agent planning that respects the distributed structure of the system, preserving agents privacy (Brafman 2015). According to their experiments, their system showed the best performance in regards to planning time and communication, as well as the quality of the solution (in most cases), when compared to other offline multi-agent planning systems.

FLAP (Sapena, Onaindia, and Torreño 2015) is a hybrid planner that combines partial-order plans with forward search. The planner uses a parallel search technique that diversifies the search. FLAP exploits delaying commitment to the order in which actions are applicable. This is done to achieve flexibility, reducing the need of backtracking and minimising the length of the plans by promoting the parallel execution of actions. These changes come at an increase in computational cost, but it allows FLAP to solve more problems than other partial-order planner.

In (Clement, Durfee, and Barrett 2007), multi-agent planning algorithms and heuristics are proposed to exploit summary information during the coordination stage, in order to speed up planning time. The authors claim that by associating summary information with plans' abstract operators, it can ensure plan correctness, even in multi-agent planning, while still gaining efficiency and not leading to incorrect plans. The key idea is to annotate each abstract operator with summary information about all of its potential needs and effects. This process often resulted in an exponential reduction in planning time compared to a flat representation. Their approach depends on some specific conditions and assumptions, and therefore cannot be used in all domains.

Multi-Agent Planning Language (MAPL) is proposed in (Brenner and Nebel 2009), for modelling MAP domains in online planning. Plans expressed in this language interleave planning, acting, sensing, and communicating. 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 private knowledge and are coordinated through the organisation via social laws. Their interleave mechanism could be used in DOMAPS to shorten the time between planning and execution.

Kovacs proposed an extension for PDDL3.1 that enables the description of multi-agent planning problems (Kovacs 2012) in PDDL. It copes with many of the already discussed open problems in multi-agent planning, such as the exponential increase of the number of actions, but it also approaches new problems such as the constructive and destructive synergies of concurrent actions. Although only the formalism is provided (it is not yet supported by any planner), the ideas expressed by Kovacs are enticing, making it an interesting candidate to add to DOMAPS planning formalisms.

Markov Decision Processes (MDP) are often used when dealing with non-deterministic worlds to coordinate agents during planning. These methods can also be extended to deal with partially observable environments, known as Partially Observable Markov Decision Processes (POMDP). For example, in (Wu, Zilberstein, and Chen 2011), the authors use an online algorithm for planning under uncertainty in multi-agent settings modelled as decentralised POMDPs, requiring little to no communication. While in (Brafman, Shani, and Zilberstein 2013), qualitative decentralised POMDP is proposed as a a qualitative, propositional model for multi-agent planning under uncertainty with partial observability.

In multi-agent POMDPs, the action and observation space grows exponentially with the number of agents. In (Amato and Oliehoek 2015), a scalable approach based on sample-based planning and factored value functions that exploits multi-agent structure to produce a scalable method for Monte Carlo tree search for POMDPs. They formalise a team of agents as a multi-agent POMDP, and introduce an online planner that uses factored statistics and factored trees to reduce the number of joint actions and the number of joint histories considered. Adding these approaches to DOMAPS could allow us to consider domains with non-deterministic actions and partially-observable environments.

Plan repair is the re-use of fragments of an old plan, and can be used to effectively simplify the coordination stage of planning. The authors of (Komenda, Novak, and Pechoucek 2014), 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. They also describe three algorithms for domain-independent multi-agent plan repair. At the moment, the re-planning in DOMAS works by restarting the planning process from scratch (but with updated information about the world). Integrating these plan repair algorithms could provide some important improvements to re-planning in DOMAPS.

Techniques for solving multi-agent pathfinding problems are also becoming more common. These problems relate to finding paths from start to goal positions for all agents, while avoiding collisions. For example, in (Sharon et al. 2015), a new multi-agent pathfinding algorithm is presented. The conflict based search algorithm works on two levels: at the high level the search is performed on a conflict tree containing the conflicts between individual agents; and at the low level single-agent searches are performed to satisfy the constraints found at the high level. This conflict tree could be useful for mapping the conflicts for the coordination mechanisms of DOMAPS. Combining path planning with task planning, such as suggested by (Srivastava et al. 2014), could also be useful in order to run real world scenarios with multiple robots.

7 Conclusion

In this paper, we described the design of the Distributed Online Multi-Agent Planning System (DOMAPS). Specifying each of its main components: *i)* the planning formalism – we introduced the MA-HTN formalism, a multi-agent variation of the traditional single-agent HTN formalism; *ii)* the goal allocation mechanism – by using a contract net protocol, the agents that participate in the planning stage can pre-select the goals that they believe to be more appropriate to them, this pre-planning can cut the planning time considerably in domains with heterogeneous agents and varied goals; *iii)* the 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 in Jason agents;

iv) the coordination mechanism – employment of social laws to coordinate the agents during run-time in order to avoid possible conflicts made during planning.

Initial experiments and experience with DOMAPS has presented enough positive incentives to pursue solutions for the limitations and to provide improvements for the framework overall, for example by adding new approaches to each component. The performance of our coordination mechanism is limited to the designer ability of detecting conflicts and formulating suitable social laws, similarly to the way that HTN depends on good methods.

Acknowledgments

We are grateful for the support given by CAPES and by CNPq (grant number 308095/2012-0).

References

Amato, C., and Oliehoek, F. A. 2015. Scalable planning and learning for multiagent pomdps. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 1995–2002.

Boissier, O.; Bordini, R. H.; Hübner, J. F.; Ricci, A.; and Santi, A. 2011. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*.

Bordini, R. H.; Wooldridge, M.; and Hübner, J. F. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons.

Brafman, R. I.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Proceedings of the Twenty-Seventh Conference on Artificial Intelligence*, 130–137.

Brafman, R. I. 2015. A privacy preserving algorithm for multi-agent planning and search. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1530–1536.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

Cardoso, R. C., and Bordini, R. H. 2016. A multi-agent extension of hierarchical task network. 10th Workshop-School on Agents, Environments, and Applications (WESAAC).

Clement, B. J.; Durfee, E. H.; and Barrett, A. C. 2007. Abstract reasoning for planning and coordination. *Journal of Artificial Intelligence Research (JAIR)* 28:453–515.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *21st European Conf. on Artificial Intelligence (ECAI'14).*

desJardins, M. E.; Durfee, E. H.; Ortiz, C. L.; and Wolverton, M. J. 1999. A survey of research in distributed, continual planning. *AI Magazine* 20(4).

Durfee, E. H., and Zilberstein, S. 2013. Multiagent planning, control, and execution. In Weiss, G., ed., *Multiagent Systems 2nd Edition*. MIT Press. chapter 11, 485–545.

Hübner, J. F.; Sichman, J. S.; and Boissier, O. 2007. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Software Engineering* 1(3/4):370–395.

Komenda, A.; Novak, P.; and Pechoucek, M. 2014. Domain-independent multi-agent plan repair. *Journal of Network and Computer Applications* 37:76 – 88.

Kovacs, D. L. 2012. A multi-agent extension of pddl3.1. In *Proceedings of the 3rd Workshop on the International Planning Competition (IPC)*, ICAPS-2012, 19–27.

Meneguzzi, F., and De Silva, L. 2013. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review* FirstView:1–44.

Nau, D.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Nissim, R., and Brafman, R. I. 2014. Distributed heuristic forward search for multi-agent planning. *J. Artif. Intell. Res. (JAIR)* 51:293–332.

Omicini, A.; Ricci, A.; and Viroli, M. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17(3):432–456.

Ricci, A.; Piunti, M.; Viroli, M.; and Omicini, A. 2009. Environment programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*, Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer. chapter 8, 259–288.

Sapena, O.; Onaindia, E.; and Torreño, A. 2015. FLAP: applying least-commitment in forward-chaining planning. *AI Commun.* 28(1):5–20.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40 – 66.

Shoham, Y., and Tennenholtz, M. 1995. On social laws for artificial agent societies: Off-line design. *Artif. Intell.* 73(1-2):231–252.

Singh, M., and Chopra, A. 2010. Programming multiagent systems without programming agents. In Braubach, L.; Briot, J.-P.; and Thangarajah, J., eds., *Programming Multi-Agent Systems*, volume 5919 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 1–14.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* 29(12):1104–1113.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA).*

Weerdt, M. D., and Clement, B. J. 2009. Introduction to Planning in Multiagent Systems. *Multiagent Grid Syst.* 5(4):345–355.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* 175(2):487–511.