UNIVERSIDAD POLITÉCNICA DE VALENCIA

Depto. Sistemas Informáticos y Computación

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

TRABAJO DE FIN DE MÁSTER

Design and implementation of a Multi-Agent Planning system

Autor: Alejandro Torreño Lerma

Dirigido por:
Dra. Eva Onaindía de la Rivaherrera
Dr. Óscar Sapena Vercher

Grupo de Tecnología Informática - Inteligencia Artificial
Group of Reasoning on Planning and Scheduling (GRPS-AI)
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n
46022 Valencia, Spain
8 de Septiembre de 2011

A mis padres, Mª Amparo y Mariano.

A Borja y Carlos.

Contents

Acknowledgements

This work has been partly supported by the Spanish MICINN under projects TIN2008-06701-C03-01 and Consolider Ingenio 2010 CSD2007-00022, and the Valencian Prometeo project 2008/051.

Chapter 1

Introduction

Planning is the art of building control algorithms that synthesize a course of action taken to achieve a desired set of goals from an initial situation. The mainstream in practical planning is that of using utility functions, which are usually called heuristics, to evaluate goals, and choices of action or states on the basis of their expected utility to the planning agent [GNT04].

Multi-Agent Planning (MAP) generalizes the problem of planning in domains where several agents plan and act together, and have to share resources, activities, and goals. In a cooperative approach, where the agents are assumed to be cooperative, the emphasis is placed on how planning can be extended to a distributed environment. The planning agents of a MAP task exchange information about their plans, which they iteratively refine and revise until they fit together [dDOW99]. Typically, research in MAP has been more concerned with the design of distributed planning architectures, mechanisms for plan coordination, or solutions for merging the resulting local plans of agents into a global plan [Dur99, CDB05, dWtMW05]. Unlike these approaches, which emphasize the problem of controlling and coordinating a posteriori local plans of independent agents, we propose a MAP model that allows agents to jointly devise a global shared plan and carry out collective actions. In our proposal, agents are able to plan concurrent actions through the adoption of the Partial-Order Planning (POP) paradigm [BW94].

We consider that MAP is about the construction of a course of action (plan) among several heterogeneous agents who have different capabilities and different views of the world. In this work, we propose a MAP model whose final objective is to form a competent global plan through the composition of the individual plans proposed by the participants. Planning agents will accomplish a MAP problem through the adoption of a refinement planning approach, whereby agents propose successive refinements to a base plan until a consistent joint plan that solves the problem goals is obtained. This

is achieved while maintaining a distributed vision of the world, which guarantees that the agents keep their private information.

However, the focus of this dissertation is not only to present a novel MAP model, but also the design and the first functional implementation of a MAP system based on the aforementioned model, which is devised to cope with the problem of cooperative planning, i.e. multiple agents working together towards the construction of a joint plan that meets the global goals of a planning problem. The model builds upon the concept of refinement planning whereby agents propose successive refinements to a base plan until a consistent joint plan that solves the problem goals is obtained. Hence, we present a description of the the main components of the system along with implementation details and experimental results.

The present document is organized as follows: chapter 2 presents some background on the main topics related to this work; chapter 3 outlines the theoretical model upon which the implemented system is based; chapter 4 summarizes the design of our MAP system, including a description of its main components and its functionalities; chapter 5 gives some implementation details about our MAP system and presents the experimental results, and finally, chapter 6 concludes and summarizes our future lines of research.

Chapter 2

Background

The present chapter introduces a state of the art on the main topics related to this work: section 2.1 presents an introduction on the topic of classical planning; section 2.2 summarizes the topic of MAP; section 2.3 introduces the Partial-Order Planning paradigm, and section 2.4 details the features and evolution of the most relevant planning description languages.

2.1 Single-Agent Classical Planning

The classical planning problem is defined as follows [Wel99]: given a description of the initial state of the world (in some formal language, usually propositional logic) denoted by $\mathcal{I}$, a description of the agent's goal (i.e., what behaviour is desired), denoted by $\mathcal{G}$, and a description of the possible (atomic) actions that can be performed, denoted by $\mathcal{A}$, and modeled as state transformation functions, find a plan, i.e., a set of actions that transforms the initial state into a state in which the agent's goal holds. Hence, classical planning can be seen as a search process in which a single agent synthesizes a set of actions that allows it to reach its objectives. Classical planners face two important issues: the definition of robust and expressive languages for the modeling of actions and change, and organizing the search for plans capable of achieving the goals, i.e. the development of efficient problem-solving techniques.

Over the years, research on classical planning has been shifting focus between different problem-solving paradigms. The motivation to introducing a new planning paradigm is to find better search techniques and representations of the planning problem. Planning paradigms can be classified according to the following concepts [Sap05]:

- **Node representation:** A planner can perform a state-based search,

where a node in the search tree represents a particular situation, or a plan-based search, in which nodes represent plans.

- **Chaining type:** A forward-chaining planner departs from the initial state to solve the problem's goals. Backward-chaining planners perform the search process on the opposite direction, which reduces the branching of the search tree, as the search is guided by the goals.

- **Plan representation:** Total-order planners build partial solutions as completely ordered sequences of actions, while partial-order planners establish partial-order constraints between the actions of each partial solution.

The early planners performed a state-based search, modeling it in terms of exploration of the space of world states, with the transitions between states being effected by the actions. Later, the need of manipulating partial plans during search became clear, which leaded to the design of plan-space search algorithms that explore the space of partial plans [PW92, BW94, Wel94]. The Partial-Order Planning (POP) approach proceeds by refining the partial plans through the addition of actions, causal links and ordering constraints. Other approaches apply a different type of refinement by replacing abstract actions by plan fragments that are capable of carrying out those actions. This is known as Hierarchical Task Planning (HTN) [EHN94], in which a planning problem is decomposed into an ordered set of abstraction levels and actions are successively refined such that the remaining actions can be done by a single agent.

During the last few years, we have seen a number of promising approaches that significantly increase the efficiency of planning systems. Most prominent among these are GRAPHPLAN [BF97], SATPLAN [KS96] and heuristic-search planning [BG01]. GRAPHPLAN and SATPLAN work both in stages by building suitable structures and then searching them for solutions. In GRAPHPLAN, the structure is a graph, while in SATPLAN it is a set of clauses. In the heuristic planning approach, a heuristic function is derived from the specification of the planning instance and used for guiding the search through the state space. This approach turned out to be very successful as demonstrated by the FF planner [HN01].

As a whole, the planning community is still very active on the task of finding efficient search techniques for single-agent classical planning, in particular in designing domain-independent techniques that allow fully automated planning systems to improve their performance. While classical planning model has driven the majority of research in planning, more and more attention is also being paid on planning in environments that are stochastic, dynamic and

partially observable (and thus do not satisfy the classical planning assumptions). The ultimate goal is to able to tackle real-world planning problems and hence it becomes necessary to address all these issues as well as others like handling uncertainty in the domain, monitoring and execution of a plan, distributed planning, planning in real-time environments, etc.

2.2 Multi-Agent Planning

The term Multi-Agent Planning (MAP) refers to the problem of planning in domains where several independent entities (agents) plan and act together. MAP is concerned with planning *by* multiple agents, i.e. distributed planning, and planning *for* multiple agents, i.e. multi-agent execution. It can involve agents planning for a common goal, an agent coordinating the plans (plan merging) or planning of others, or agents refining their own plans while negotiating over tasks or resources [Cle05]. Planning agents may collaborate in order to reach common goals or they may act selfishly in order to satisfy their own private goals. Hence, the nature of MAP problems introduces some challenges that are not present in classical planning, like how the agents interact to develop a distributed plan, how planning information is distributed, which information about the problem they own, how common goals are distributed, or how private goals are treated.

MAP problems and solutions for multi-agent environments differ considerably from their single-agent counterparts. Domain knowledge is usually distributed among agents, so agents typically work with an incompletely known domain and do not likely have access to all the information about the initial state of the MAP problem. Some MAP approaches allow the initial state of a planning problem to be incomplete [BN09], and address the problem of planning with incomplete information by following a conformant planning approach [HB06]. Other approaches adopt the construction of complete knowledge bases, which enables plan proposal generation through planning with complete information [BRR10].

The problem of distributed planning has been addressed from different perspectives: one approach has begun with a focus on planning and how it can be extended into a distributed environment, and the other approach has begun with an emphasis on the problem of controlling and coordinating the actions of multiple agents in a shared environment [dDOW99]. The first approach is more related to the problem of planning by multiple agents, and it is the view usually adopted by the planning community, whilst the second approach is more concerned about the problem of planning for multiple agents (acting), and it is closely related to research in multi-agent systems.

Nevertheless, a number of works share characteristics from both approaches, particularly systems with self-motivated agents that must cooperate at some points towards a common objective. Therefore, a MAP problem can be approached from different directions depending on whether the plan can be distributed among a variety of execution systems and whether the planning process should be distributed. This gives rises to a centralized/distributed planning for centralized/distributed plans [Dur99].

The two keys aspects in a MAP problem are the planning activity by which the participating entities should develop a course of action to attain the problem goals, and a coordination process that will combine the partial solutions into a single competent plan that solves the problem. In general, the following steps can be distinguished in the process of solving a MAP problem [Dur99]: 1) global goal refinement, 2) task allocation, 3) coordination before planning, 4) individual planning, 5) coordination after planning, and 6) plan execution. Not all the steps need to be included. For instance, steps 1 and 2 are not needed if there are no common goals. Also, some approaches combine different phases. For example, agents can already coordinate their plans while constructing their plans (combination of phase 4 and 5), or postpone coordination until the execution phase (combination of phase 5 and 6), as in the case of continuous planning, an ongoing and dynamic process in which planning and execution are interleaved [Mye99].

The next subsection presents some of the most relevant coordination mechanisms in multi-agent planning.

2.2.1 Coordination in MAP

As outlined in the previous section, one of the basic steps in the process of solving a MAP task focuses on coordinating the agents' activity in order to find a solution plan. This task can be carried out at different stages of the MAP process [dWtMW05]. Some approaches perform the coordination prior to the actual planning process, while others allow the agents to perform an individual planning and focus on merging the resulting plans on a joint solution. Finally, some works combine the planning and coordination activities by interleaving them.

Most approaches to MAP emphasize the problem of distributing the planning process among several planning agents and merging the resulting local plans to come up with a joint plan that meets the global goals. Some works address this problem under the continual planning approach [dDOW99], in which planning and execution are interleaved and coordination of local plans is achieved via synchronization of agents at execution time [BN09, CB03]. Other approaches assume the existence of self-interested agents, and define

MAP as the problem of finding a plan for each agent that achieves its private goals, such that these plans together are coordinated and the global goals are met as well [dWtMW05]. Under this perspective, the emphasis is on how to manage the interdependencies between the agents' plans and choose the mechanisms to solve the coordination problem. Some frameworks use, however, a pre-planning coordination approach, which establishes a set of coordination constraints prior to the planning process, in order to guarantee that the resulting plans will be conflict-free.

There is a large body of research focused on plan merging methods aimed at the construction of a joint plan. One of the most well-known approaches to coordination of plans is the Partial Global Planning (PGP) framework [DL91]. In PGP, agents communicate their local plans to the rest of agents, which in turn merge this information into their own partial global plan and try to improve it. Such an improved plan is shown to the other agents, who might accept, reject, or modify it. In [CDB05] authors propose an extension of the Partial-Order Planning (POP) paradigm to the multi-agent case such that agents can mutually benefit by coordinating their plans and avoiding duplicating effort. Some approaches even propose algorithms to deal with insincere agents and to interleave planning, coordination, and execution [ER96].

Some MAP frameworks, however, follow a pre-planning coordination approach, by which they establish a set of coordination constraints prior to the planning process, in order to guarantee that the resulting plans will be conflict-free. Some of these approaches introduce a set of constraints called social laws [ST95, YNH92], that must be followed by the agents. These restrictions make the plan merging process easier, and guarantee that the resulting plans will be conflict-free. Other pre-planning approaches try to figure out the interactions among the planning agents' tasks beforehand [TPW03].

Since a MAP problem can be viewed as the problem of having individually-motivated agents cooperating for a common objective (typically the construction of a competent overall plan), negotiation underpins attempts to cooperate and coordinate, and is required both when the agents are self-interested and when they are cooperative [JFL$^+$01]. We can also find several works in the literature on negotiation as a tool for coordinating cooperative agents in distributed planning [LL92, MLB92].

Another recent research trend focuses on the use of argumentation for coordination in MAP. The TRAINS system [ASF$^+$95], a research platform that considers the problem of representing plans for mixed-initiative planning, was the first framework on introducing this approach. The work in [BRR09, BRR10] proposes an argumentation-based approach for coordinating several agents, aimed at discussing the validity of the plan proposals.

Other approaches take into account the communication needs that arise when plans are being executed [TNP09].

2.3 Partial-Order Planning

In Total-Order Planning approaches, the plan's actions are obtained in the same order in which they are executed. This way, if a planner chooses a wrong action, it will have to introduce another action to undo the effects of the first one. As opposite to these models, the *POCL* (Partial Order Causal Link) Partial-Order Planning (*POP*) paradigm [BW94] introduces a more flexible approach, establishing partial order relations between the actions in the plan instead of enforcing a concrete order among them. POP-based planners 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 it. This strategy based on deferring decisions during the planning search is known as least commitment strategy [Wel94].

Instead of performing a state-based search, POP models adopt a plan-based search approach. Hence, a POP addresses the process of building a search tree in which each node represents a different partial plan, without maintaining the notion of planning state. POP is also classified into the backward-chaining search approaches, since it begins the search by satisfying the problem goals, and builds the plan backwards. In conclusion, POP can be considered a plan-based, backward-chaining search process.

Partial plans contain a set of *steps* [Wel94], which are the representations of planning actions within the partial plan. Hence, each step is composed by a set of preconditions and effects, except for two fictitious steps, included in every partial plan, that represent the initial state and the goals. Steps are partially ordered through a set of *ordering constraints*. A partial-order constraint indicates only a precedence relationship between two steps. Hence, partial-order constraints allow other steps to be ordered between the steps that form the constraint.

An *open goal* is a precondition which is not supported by a *causal link*. A causal link indicates that a step supports a precondition of another step, by having an effect that matches with the involved precondition. The introduction of causal links in a partial plan may trigger the appearance of *threats* as a side effect. A threat occurs when there is a step that can be ordered between both steps of the causal link, and it has an effect that is complementary to the precondition enforced by the causal link. Both open goals and threats are referred to as the plan's *flaws*. Hence, the POP process focuses

on solving all the flaws of a partial-order plan through the introduction of causal links and ordering constraints

POP became the focus of the planning research in the 1990s. However, it faces an important challenge that has caused planning research to put the emphasis on state-based approaches. This issue is related to the lack of scalability of the POP algorithms. The problem of defining a high quality heuristic to guide the POP plan-space search process remains unsolved. Currently, there is not a single POP heuristic that can make this process competitive against the latest state-based planning frameworks; even the latest efforts, although improving the POP performance [NK01], are not efficient enough to be competitive against state-based approaches. For this reason, POP has been discontinued by the vast majority of the planning community, in favor to more recent state-based approaches. However, the growing interest on MAP frameworks has made POP gaining relevance again, since its flexibility makes it a good alternative to tackle multi-agent planning problems.

2.4 Planning languages

One of the main issues that have been addressed by the planning community is the representation problem. The use of a good planning language is one of the key aspects of an efficient planning process. Since the 1970s, most of the planning works have been widely influenced by the *STRIPS* language [FN71] as it effectively solves the frame problem [MH69], and it supports divide-and-conquer strategies [Gef00]. This section describes briefly this language and its most relevant extensions: *ADL* [Ped89] and *PDDL* [McD00].

2.4.1 STRIPS

The *STRIPS* (STanford Research Institute Problem Solver [FN71]) language was developed on the earlier 1970s, as a planning system for the robot *Shakey*. *STRIPS* proposes a compact and simple model to specify planning domains.

The *STRIPS* representation includes various restrictions that limit its expressivity, making it difficult to describe some real problems [RN03]. As a result, many extensions to *STRIPS* have been developed over the past years, enriching its expressiveness and simplifying the definition of planning domains. The main restrictions introduced by *STRIPS* can be summarized as follows:

- **Only positive literals:** It is not possible to negate literals explicitly in the preconditions of the actions or in the initial state. The unmentioned

literals are assumed to be *false*. Further extensions of the *STRIPS* language will introduce the possibility of using negated literals, and will assume the unmentioned literals to be *unknown*.

- **Delete effects meaning:** A negated effect $\neg P$ implies the deletion of P. Further *STRIPS*' extensions consider that an effect $\neg P$ means the deletion of P and the addition of $\neg P$.
- **Only conjunctions in preconditions, effects and goals:** Unlike its extensions, STRIPS does not allow the use of disjunctive preconditions, effects and goals. The states of the problem are described by using only conjunctions of positive literals.
- **Only grounded literals in goals:** *STRIPS* only allows the use of grounded literals in the description of the goals, while its successors consider the possibility of using ungrounded facts.
- **Lack of types support:** The possibility of defining a type hierarchy is one of the main additions of the *STRIPS*' extensions, simplifying and making more readable the description of planning domains.
- **Lack of equality support:** As opposite to its extensions, *STRIPS* does not allow the definition of equality restrictions ($a = b$).

Apart from addressing these restrictions, the *STRIPS*' extensions augment the language model by introducing features like numerical expressions and time management, as shown in the following sections.

2.4.2 ADL

Although the *STRIPS* language is quite restrictive, it can be easily extended to satisfy the requirements of complex domains. One of the main extensions developed is the (*Action Description Language* (*ADL*) [Ped89]). *ADL* uses an algebraic model to define the states of the world, which makes it more expressive than *STRIPS*. *ADL* improves *STRIPS* by incorporating the following features:

- **Types:** *ADL* allows to set a type to the objects of the problem and the parameters of the preconditions, which improves the comprehension of the problems and simplifies the planning domains.
- **Negated goals and preconditions:** The preconditions of an *ADL* operator can include negated atomic formulas. Moreover, it is possible

to specify goals as negated literals to represent facts that are not desired in an objective state.

- **Disjunctive preconditions:** An *ADL* precondition can be a disjunction of atomic formulas.
- **Quantified formulas:** The preconditions in *ADL* can include both existentially and universally quantified formulas.
- **Equality restrictions:** *ADL* introduces a new type of atomic formula in the preconditions, that is satisfied when its two arguments are equal. This predicate, named *Equality* allows the comparison of variable symbols within an operator.
- **Conditional effects:** These effects are only effective if the specified condition is satisfied in the state in which the action is applied. Conditional effects are usually placed on the quantified formulas.

The extensions introduced by *ADL* improve considerably the expressivity of *STRIPS*, allowing the designer to represent a larger number of situations. It is possible to take advantage of these new features to improve the efficiency of many planning systems [KNHD97].

2.4.3 PDDL

Although *ADL* is one of the most popular *STRIPS* extensions, many other have been developed. For instance, *FStrips* (*Functional STRIPS* [Gef00] is a first order language (without quantification), that uses constants, functions and relational symbols (although it does not use variable symbols), improving the language's expressivity. However, the most successful extension of *STRIPS* has been *PDDL* (the Planning Domain Definition Language [GHK+98]). *PDDL* was developed for the 1998 International Planning Competition [McD00], aiming to provide a common notation for modeling planning problems and evaluating the planners' results. Ever since *PDDL* was introduced, it has become the reference modeling language for the vast majority of planners.

Along with *STRIPS* and *ADL*, *PDDL* has been influenced by many other formalisms: *SIPE-2* [Wil88], *Prodigy 4.0* [BEG+92], *UCMP* [EHN94], *Unpop* [McD96] and *UCPOP* [BCF+95]. The goal pursued by *PDDL* is to express the domain's *physics*, i. e., which predicates are present, which actions can be applied, and which their effects are, without providing any additional knowledge about it. *PDDL* offers a wide set of features, from which the following ones stand out:

- ***STRIPS*-based action modeling.**
- **Conditional effects and universal quantification.**
- **Hierarchical actions:** actions can be decomposed in subactions and subgoals, which allows the designer to deal with more complex problems.
- **Domain axioms:** axioms are logical formulas that establish relations between the facts that are satisfied in a state (as opposite to actions, which define relations between successive states).
- **Security restrictions:** these constraints allow for the definition of a set of objectives that must be accomplished over all the planning process.

Most of the existing planners do not handle the full set of features introduced by *PDDL*. For simplicity, *PDDL*'s features are gathered into sets of requirements. In this way, planners can easily check if they are capable to handle a certain domain.

2.4.4 PDDL extensions

The International Planning Competition (*IPC*) [DKS+00] has become an important conductor for planning research. One of the most important outcomes of the first edition of the *IPC* was the adoption of *PDDL* as a common planning definition language [MGH+98]. Furthermore, the following editions of the event have improved the language by the introduction of new extensions. This section discusses the most relevant *PDDL* existing extensions and the main features introduced by each extension.

PDDL2.1 One of the most relevant contributions of the 2002 *IPC* (*IPC-3*) was the introduction of the first *PDDL* revision: *PDDL2.1* [FL03]. This extension is completely backwards compatible with *PDDL*, and improves its expressivity by adding time management and numeric capabilities. The most relevant improvements introduced by *PDDL2.1* are the following ones:

- **Numeric fluents:** *PDDL2.1* proposes a definitive syntax for the expression of numeric expressions, conditions and effects. Conditions on numeric expressions are always comparisons between pairs of numeric expressions. Numeric effects can be direct or relative assignments.

- **Durative actions:** *PDDL2.1* allows to specify *discrete* and *continuous* durative actions. The differences between both forms refer to how numeric values change over the interval of the action. The occurrence of an effect on a discrete durative action can be modeled at the beginning, at the end or over all its execution. Continuous durative actions, in turn, allow the specification of continuous effects.

- **Plan metrics:** *PDDL2.1* allows for the definition of an objective function or metric, which specifies the basis on which a plan will be evaluated for a particular problem. Plan metrics can be defined by using a primitive numeric expression specified in the domain, or through the special variable `total-time`, which refers to the temporal span of the entire plan.

For the ease of reference, the features of *PDDL2.1* have been grouped in four different levels of increasing expressive power. Level 1 encloses the propositional and *ADL* levels of the previous *PDDL* version; level 2 establishes a syntax to handle numeric expressions (fluents); level 3 manages the discrete durative actions, and level 4 allows the durative actions to have continuous effects.

Finally fifth level has been added by a more recent extension called *PDDL+*. It allows to model efficiently the appearance of events and processes (activities that cause continuous changes on the values of the numerical expressions while they last). Although this classification includes five levels, only the first three levels have been used by current state-of-the-art planners, since the planning technology is not sufficiently advanced to handle the additional complexities introduced by the last two levels.

PDDL2.2 The fourth *IPC* (*IPC-4*), run in 2004, introduced a new revision to *PDDL*, *PDDL2.2* [Ede03]. As the extensions introduced in *PDDL2.1* still provide major challenges to the planning community, the number of changes added by *PDDL2.2* is relatively moderate. Since the first three levels of PDDL2.1 are considered an agreed fundament, *PDDL2.2* inherits that structure, adding a set of new features on top of it. The main improvements added by *PDDL2.2* are the following ones:

- **Derived axioms:** *PDDL2.2* offers the possibility to define predicates that are not affected by any of the actions defined in the domain. Instead, the truth values of predicates are derived by a set of rules of the form $iff(x)thenP(x)$. Derived predicates are included into the three levels of the language.

- **Timed initial literals:** The language includes the possibility to define literals that will become *true* or *false* at established time points, independently of the actions selected by the planner to execute. Timed initial literals are thus a simple way of expressing deterministic unconditional exogenous events. These literals can be used at the level 3 of the language.

PDDL3.0 *PDDL3.0* [GL05] was developed for the 2006 *IPC* (*IPC-6*). Unlike the previous versions, *PDDL3.0* emphasizes the importance of plan quality. The previous versions allowed the user to express some criteria on plan quality, and relatively complex plan metrics. In order to improve the language power to express conditions on the plan quality, *PDDL3.0* has introduced a whole set of new features, that can be synthesized as follows:

- **State trajectory constraints:** These constraints assert conditions that must be met by the entire sequence of states visited during the execution of a plan. They are expressed through temporal modal operators over predicates.
- **Soft constraints, preferences and plan quality:** *PDDL3.0* allows to express conditions that are preferred to see satisfied on the trajectory generated by a plan, even though these preferred constraints do not have to be necessarily satisfied. If the domain includes multiple soft constraints, it is possible to establish which of them should take priority in case of conflict. *PDDL3.0* provides also constructs to describe how the satisfaction of these constraints (or lack of it) affects the quality of a plan.

PDDL3.1 The latest *PDDL* version, *PDDL3.1* [Kov11], was introduced in the context of the 2008 *IPC*. The main purpose of this extension was to enrich the language with SAS+-like [BN95] problem representations. However, SAS+ only allows using functional fluents in very limited ways since it does not allow nesting and only allow comparisons and assignment to constants. Hence, *PDDL3.1* introduces object fluents, which are state variables which are neither binary (true/false) nor numeric (real-valued), but instead map to a finite domain, a more flexible solution inspired by the *Functional Strips* formalism [Gef00]. In addition, *PDDL3.1* allows to specify numeric action costs in language fragments that don't normally permit numerical features. The main additions provided by the language can be summarized as follows:

- **Object fluents:** These state variables map to a finite domain of objects, defined by an object type. Only one object of the domain can

be assigned to the variable at each planning state. Since they can be employed in the preconditions and effects of the actions, object fluents can be used in conjunction with traditional predicates or even replace them in the planning models.

- **Action costs:** This feature allows the user to assign a numeric, non-negative cost to the actions in the domain. This way, action costs play an important part in determining plan quality.

Chapter 3

Multi-Agent Planning theoretical framework

This chapter presents the Multi-Agent Planning (MAP) framework in which our MAP system implementation is based. It also describes the procedure followed by the agents for building and exchanging plans among them. The first section of the chapter models the main components of a MAP task, while the second one outlines the refinement planning process.

We consider that MAP refers to multiple agents planning and acting collaboratively. More specifically, agents interact to design a plan that none of them could have generated individually in most cases. During the plan construction, agents keep in mind that the devised plan will be jointly executed by themselves such that they collectively achieve their individual and common goals.

As outlined in section 2.2, the domain information in MAP is usually distributed among planning agents, so they are not likely to have access to all the information about the problem's initial state. In our model, the incomplete information of an agent only concerns its partial view of the domain and as well as the ignorance about the knowledge possessed by the rest of participants, but the initial state of the problem is known among all of the agents. Thus, the initial state may not be fully known to each agent as well as the capabilities or specialized expertise of an agent may be also unknown to the rest of agents. On the other hand, we define a set of common goals, known by all of the agents, and, possibly, a set of individual goals specified as soft constraints [GL06]. Individual goals are only known to each particular agent and, unlike common goals, the agent is not committed to satisfy them.

In our approach, the planning model of an agent extends the classical STRIPS-like planning model, allowing agents to represent their partial view

of the world and adopting the open world assumption [Rei87] as opposite to STRIPS-like models. In practice, each agent uses the classical single-agent planning language *PDDL* [MGH+98], more precisely the features included in *PDDL3.1* [Kov11] plus some additions to deal with specific MAP requirements.

Informally speaking, we define a MAP problem as follows: given a description of the initial state, a set of global goals, a set of (at least two) agents, and for each agent a set of its capabilities (the actions they can perform) and (probably) its private goals, find a plan for each agent, such that these plans together are coordinated and the problem's global goals are met [dWtMW05]. In our approach, by MAP we denote a planning process distributed across several planning/executing agents who devise a joint, non-linear plan which will be later executed by the same agents. We assume that agents are specifically designed to be cooperative but can also have their own private goals. In our view of MAP, agent's decisions must not only be derivative from the collective goals but also from the other agents' decisions.

3.1 Specification of a MAP task

Definition 1. *(**MAP task**) A **MAP task** is a tuple* $\mathcal{T} = \langle \mathcal{AG}, \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{F} \rangle$*:*

- $\mathcal{AG}$ *is a finite non-empty set of planning agents.*
- $\mathcal{O}$ *is a finite set of objects, that model the elements of the planning domain over which the planning actions can act.*
- $\mathcal{V}$ *is a finite set of state variables that model the states of the world. Each state variable* $v \in \mathcal{V}$ *is mapped to a finite domain of mutually exclusive values* $\mathcal{D}_v$*. Each value in a state variables's domain corresponds to an object of the planning domain i.e.* $\forall v \in \mathcal{V}$, $\mathcal{D}_v \subseteq \mathcal{O}$*.*
- $\mathcal{A}$ *is the set of deterministic actions of the agents' models.*
- $\mathcal{I}$ *is a set of assigned state variables that models the initial state of the MAP task* $\mathcal{T}$*.*
- $\mathcal{G}$ *is is a set of state variables that represent the problem's common or global goals that must be achieved by the agents in order to fulfill the MAP task.*
- $\mathcal{F}$ *is a global utility function used by the agents to select a plan when several choices are available.*

When assigned to a certain value of their domain, the state variables act as propositions in propositional planning. We regard assigned state variables and actions as atomic, just as in propositional planning. Despite the fact that not every state variable is known to all the agents, they must share a common language to allow communication between them. To denote the actions, goals, etc. of an agent $i \in \mathcal{AG}$ we will use the superscript notation x^i for any such aspect x.

Definition 2. *(**Agent's bases**) An agent $i \in \mathcal{AG}$ is equipped with **three bases** $\langle \mathcal{I}^i, \mathcal{A}^i, \mathcal{PG}^i \rangle$:*

- *$\mathcal{I}^i$ a set of literals which models the agent's knowledge about the problem's initial state.*
- *$\mathcal{A}^i$, a base of planning rules or actions which represents the agent's capabilities.*
- *$\mathcal{PG}^i$, a (possibly empty) set of private goals.*

The information on the initial state $\mathcal{I}$ is distributed among agents, which have a partial knowledge about $\mathcal{I}$. We also assume that agents' knowledge about $\mathcal{I}$ is fully consistent i.e. there is not contradictory information among agents. Consequently, the initial state of a planning task $\mathcal{I}$ can be defined as $\mathcal{I} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{I}^i$.

Definition 3. *(**Literal**) A **literal** $(v = d)$, where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$, is a state variable v assigned to a certain object d, which is part of the variable's domain $\mathcal{D}_v$. A literal can also be a negated assigned state variable $\sim (v = d)$, where $\sim$ represents the strong negation.*

The initial state of an agent, $\mathcal{I}^i$, is modeled through a set of literals. As stated in the previous definition, a literal is a state variable assigned to a concrete value of its domain. Since values of a variable's domain are mutually exclusive, the assignment of a variable to a value discards the rest of values of its domain. Literals are either assigned state variables or negated assigned state variables, which allows agents to represent explicitly both the true and false information, rather than applying negation by failure, as the STRIPS-like models. Hence, our model adopts the open world assumption, considering ignorance by failure, i.e. the information which is not represented in the initial state is unknown to the agents.

Each agent $i \in \mathcal{AG}$ is associated with a set $\mathcal{A}^i$ of possible actions such that the set of actions of a planning task is defined as $\mathcal{A} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{A}^i$. An action $\alpha \in \mathcal{A}^i$ denotes that agent i has the capability expressed in the action α (i is the owner of α). If α is planned to form part of the final plan then i is also the agent responsible of executing α.

Definition 4. *(**Planning rule or action**) A **planning rule or action** $\alpha \in \mathcal{A}$ is a tuple $\langle PRE, EFF \rangle$. PRE is a set of literals representing the preconditions of α, and EFF is a consistent set of literals representing the consequences of executing α. We will denote an action α as follows: $(p_1, p_2, \ldots, p_n \Rightarrow e_1, e_2, \ldots, e_m)$, where $\{p_1, \ldots, p_n\}$ denote the preconditions and $\{e_1, \ldots, e_m\}$ the effects of α, respectively.*

An action α may belong to the set of possible actions of different agents. Thus, it can be the case that $\alpha \in \mathcal{A}^i$ and $\alpha \in \mathcal{A}^j$, being $i \neq j$. We will use PRE and EFF as the precondition function and effect function, respectively, for an action α: $PRE(\alpha) = \{p_1, \ldots, p_n\}$, $EFF(\alpha) = \{e_1, \ldots, e_m\}$. The set PRE mentions the literals that must hold in a world state S for that the action α is applicable in this state. The result of executing α in S is a new world state S' as the revision of S by $EFF(\alpha)$, i.e. updating the literals in S according to the effects of α. Revision entails modifying the values of the literals in S according to $EFF(\alpha)$ in order to obtain the resulting state S'.

Additionally, actions have an associated cost. $cost(\alpha) \in \mathbb{R}_0^+$ is the cost of α in terms of the global utility function $\mathcal{F}$.

Finally, the private goals of an agent i, $\mathcal{PG}^i \subset \mathcal{V}$, are positive assigned state variables (literals) that the agent is interested in attaining. Unlike the global problem goals $\mathcal{G}$, agents are not forced to achieve their private goals, so they are encoded as soft constraints [GL06].

3.2 Refinement Planning

Our MAP model can be regarded as a multi-agent refinement planning framework, a general method based on the refinement of the set of all possible plans [Kam97]. An agent proposes a plan Π that typically enforces some open goals; then, the rest of agents collaborate on the refinement of Π by offering help in solving some open goals in Π. This way, agents cooperatively solve the MAP task by consecutively refining an initially empty partial plan.

In this context, the Partial-Order Planning (POP) paradigm [BW94] arises as a suitable approach to address refinement planning, as it is focused on solving open goals progressively, as opposite to state-based frameworks. Consequently, agents in our MAP approach plan concurrent actions through the adoption of the POP paradigm. Hence, under an idealized and static environment, the distributed plan devised among the agents would be fully executable. The notion of partial plan is at the core of our planning model. In the following, we provide some basic definitions concerning single-agent POP and its adaptation to a Multi-Agent Planning (MAP) context.

3.2.1 Single-agent Partial-Order Planning

Definition 5. *(**Partial plan**) A **partial plan** is a tuple* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$*, where:*

- $\Delta \subseteq \mathcal{A}$ *is the set of planning actions in* Π*.*
- $\mathcal{OR}$ *is a set of ordering constraints* ($\prec$) *on* Δ*.*
- $\mathcal{CL}$ *is a set of causal links over* Δ*. A causal link is of the form* $\alpha \overset{v=d}{\rightarrow} \beta$*, where* $\alpha \in \mathcal{A}^i$*,* $\beta \in \mathcal{A}^j$ *are two steps of* Δ*, likely from different agents* ($i \neq j$)*, and* ($v = d$) *is a literal such that* $v \in \mathcal{V}$*,* $d \in \mathcal{D}_v$*,* $(v = d) \in EFF(\alpha)$ *and* $(v = d) \in PRE(\beta)$*.*

This structural definition of partial plan actually represents the mapping of a plan into a directed acyclic graph, where Δ represents the nodes of the graph (actions) and $\mathcal{OR}$ and $\mathcal{CL}$ are sets of directed edges representing the required precedences of these actions and the causal links among them, respectively. Note that $\Delta = \bigcup_{i=1}^{n} \Delta^i$, where each Δ^i denotes the (possibly empty) set of actions contributed by agent i to the plan Π.

An *empty* partial plan is defined as $\Pi_0 = \langle \Delta_0, \mathcal{OR}_0, \mathcal{CL}_0 \rangle$, where Δ_0 contains two fictitious actions, the initial action α_0 and the final action α_f. α_0 and α_f are not real actions and hence they belong to $\mathcal{A}$ but not to the action set of any particular agent. $\mathcal{OR}_0$ contains the constraint $a_0 \prec a_f$ and $\mathcal{CL}_0$ is an empty set. This way, a plan Π for any given MAP task $\mathcal{T}$ will always contain the two fictitious actions such that $PRE(\alpha_0) = \emptyset$ and $EFF(\alpha_0) = \mathcal{I}$, $PRE(\alpha_f) = \mathcal{G}$, and $EFF(\alpha_f) = \emptyset$; i.e. α_0 represents the initial situation of $\mathcal{T}$, and α_f represents the global goals of $\mathcal{T}$.

Assuming that $\mathcal{G} \neq \emptyset$, an empty plan is said to be non-complete because the preconditions of α_f are not yet supported by any action. Hence, the POP search process is aimed at solving the unsupported preconditions of the plans, also called *open goals*.

Definition 6. *(**Open goal**) An **open goal** in a partial plan* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ *is defined as a literal* $(v = d)$ *such that* $v \in \mathcal{V}$*,* $d \in \mathcal{D}_v$*,* $(v = d) \in PRE(\beta)$*,* $\beta \in \Delta$*, and* $\nexists \alpha \in \Delta / \alpha \overset{v=d}{\rightarrow} \beta \in \mathcal{CL}$*.* $openGoals(\Pi)$ *denotes the set of open goals in* Π*. A partial plan is said to be **incomplete** if it has open goals. Otherwise, it is said to be **complete**.*

As the POP search process progresses, some of the causal links introduced in the partial plan can become unsafe as a result of the introduction of a new step which is not ordered with respect to the causal link. These conflicts are called *threats*.

Definition 7. *(**Threat**) A **threat** in a partial plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ represents a conflict between a step of the plan and a causal link. A step γ causes a threat over a causal link $\alpha \overset{v=d_1}{\rightarrow} \beta$ if $(v = d_2) \in EFF(\gamma)$, where $v \in \mathcal{V}$, $d_1 \in \mathcal{D}_v$, $d_2 \in \mathcal{D}_v$ and $d_1 \neq d_2$, and there is not an ordering constraint $\gamma \prec \alpha$ nor $\beta \prec \gamma$. $Threats(\Pi)$ denotes the set of threats in Π.*

A threat $t \in Threats(\Pi)$ can be solved by *promoting* or *demoting* the threatening step γ with respect to the causal link $\alpha \overset{v=d_1}{\rightarrow} \beta$ i.e. introducing an ordering constraint $\gamma \prec \alpha$ or $\beta \prec \gamma$. Threats and open goals are also known as the *flaws* of a partial-order plan. Therefore, the POP process is directed at solving the pending flaws of an initially empty partial plan, in order to reach a solution for the planning task.

3.2.2 Multi-agent Partial-Order Planning

In our MAP approach, partial plans that agents build are concurrent multi-agent plans as two different actions can now be executed concurrently by the two proposer agents. Some approaches adopt the well-known notion of concurrency as established by distributed systems and non-linear planning; that is, two actions can happen concurrently if none of them changes a pre-condition or effect of the other [BN09]. More sophisticated methods of action concurrency are introduced in [CDB05], in which the assumption that actions are instantaneous is relaxed. In [BB01], authors extend the POP algorithm to represent concurrent actions with interacting effects. Our notion of concurrency follows the one in [BB01] that considers three types of conflicting interactions among actions, which are also adopted by GraphPlan to define the mutual exclusion relations in a planning graph [BF97]. These mutual exclusion relations among actions or *mutex* can be interpreted as an adaptation of the concept of threat to a MAP context.

Definition 8. *(**Mutex actions**) Two actions $\alpha \in \mathcal{A}^i$ and $\beta \in \mathcal{A}^j$ are **mutually exclusive** (or mutex) if one of the following conditions holds:*

- *If a literal $(v = d_1) \in EFF(\alpha)$ and $(v = d_2) \in PRE(\beta)$, where $v \in \mathcal{V}$, $d_1 \in \mathcal{D}_v$, $d_2 \in \mathcal{D}_v$ and $d_1 \neq d_2$, or vice versa; i.e., if the effects of an action change the value of a state variable that the other action relies on.*

- *If a literal $(v = d_1) \in EFF(\alpha)$ and $(v = d_2) \in EFF(\beta)$, where $v \in \mathcal{V}$, $d_1 \in \mathcal{D}_v$, $d_2 \in \mathcal{D}_v$ and $d_1 \neq d_2$, or vice versa; i.e., if the effects of an action threaten the effects of the other action.*

- *If a literal* $(v = d_1) \in PRE(\alpha)$ *and* $(v = d_2) \in PRE(\beta)$*, where* $v \in \mathcal{V}$*,* $d_1 \in \mathcal{D}_v$*,* $d_2 \in \mathcal{D}_v$ *and* $d_1 \neq d_2$*, or vice versa; i.e., if the precondition list of an action is not logically consistent with the precondition list of the other action.*

According to definition 8, two *concurrent actions* are logically consistent if none of the above three conditions are met in a plan. Definition 8 is straightforwardly extended to a joint action $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$.

Our notion of multi-agent concurrent plan is an extension to the concept of a single-agent, partial plan free of threats [BW94]. Hence, a multi-agent concurrent plan can be defined as follows:

Definition 9. *(**Multi-agent concurrent plan**) Given a partial plan* $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$*, we say* Π *is a* ***multi-agent concurrent plan*** *or a* ***consistent multi-agent plan*** *if for every pair of unequal actions* $\alpha \in \mathcal{A}^i$*,* $\beta \in \mathcal{A}^j$*,* $i \neq j$*, that are unordered* ($\alpha \not\prec \beta$)*, then* α *and* β *are not mutex.*

A partial-order plan can be interpreted as a linear sequence of states. Given a plan $\Pi = \langle \Delta, \mathcal{OR}, \mathcal{CL} \rangle$ for a MAP task $\mathcal{T} = \langle \mathcal{AG}, \mathcal{O}, \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{F} \rangle$, Π can also be regarded as a chronologically ordered sequence of world states $\Pi = \{S_0, S_1, \ldots S_n\}$, where each S_i is a fully-instantiated state that results from the effects of the concurrent actions at the execution step $i - 1$. Thus $S_0 = \mathcal{I} = EFF(\alpha_0)$ is the initial state, and $\mathcal{G} \subseteq S_n$, which means that the common goals of $\mathcal{T}$ are enforced in the last state S_n of the plan.

Assuming that $\mathcal{G} \neq \emptyset$, an empty plan is said to be non-complete because the preconditions of α_f are not yet supported by any action. Hence, agents in our MAP approach focus on devising *refinement plans* that solve the open goals of the partial plans. In our framework, the refinements proposed by the planning agents must be threat and mutex free. However, they can contain open goals if the proposing agent is not able to solve them or if it considers that they would be solved more efficiently by other agents. This issue underpins the agent attempts to cooperate and collaborate towards a solution plan.

Definition 10. *(**Refinement plan**) A partial plan* $\Pi_j = \langle \Delta_j, \mathcal{OR}_j, \mathcal{CL}_j \rangle$ *is a* ***refinement*** *of another partial plan* $\Pi_i = \langle \Delta_i, \mathcal{OR}_i, \mathcal{CL}_i \rangle$ *if and only if* $\Delta_i \subseteq \Delta_j$*,* $\mathcal{OR}_i \subseteq \mathcal{OR}_j$*,* $\mathcal{CL}_i \subseteq \mathcal{CL}_j$ *and* $\exists p \in openGoals(\Pi_i)/p \notin openGoals(\Pi_j)$*.*

Therefore, a refinement can be seen as a partial plan built upon a base plan and aimed at solving at least one of its open goals.

A refinement plan Π_j actually results from the composition of Π_i, the base plan, and a *refinement step* Π', where $\Pi' = \langle \Delta', \mathcal{OR}', \mathcal{CL}' \rangle$ and $\Delta_j = \Delta_i \cup \Delta'$,

$\mathcal{OR}_j = \mathcal{OR}_i \cup \mathcal{OR}'$ and $\mathcal{CL}_j = \mathcal{CL}_i \cup \mathcal{CL}'$. We will denote the composite plan as $\Pi_j = \Pi_i \circ \Pi'$.

As previously mentioned, in our multi-agent POP model, the refinement plans proposed by the agents are always mutex and threat free but can contain open goals. This means that if the refinement step contributed by an agent brings about a mutex or a threat on the composite plan, the agent is responsible for solving it and thus suggesting a refinement that addresses the flaw. Consequently, if an agent is not capable to come up with a consistent refinement plan, then the agent refrains from suggesting such a refinement. In case of no refinements for an incomplete partial plan, we say the plan is a dead-end.

Definition 11. *(**Dead-end plan**) A plan Π_i is called a **dead-end plan** if $\exists p \in openGoals(\Pi_i)$ and there is no refinement step Π' such that $p \notin openGoals(\Pi_i \circ \Pi')$; that is, no refinement step solves the open goal p.*

Finally, we define a solution plan.

Definition 12. *(**Solution plan**) A multi-agent concurrent plan Π is a **solution plan** for a planning task $\mathcal{T}$ if Π is a consistent and complete plan, i.e. it has no threats nor mutex and $openGoals(\Pi) = \emptyset$.*

Note that we require Π to be a complete plan so it cannot have pending open goals. Consequently, the preconditions of the fictitious final action α_f will also hold thus guaranteeing that Π solves the planning task $\mathcal{T}$.

Chapter 4

Multi-Agent Planning system design

The Multi-Agent Planning (MAP) theoretical framework shown in the previous chapter has been taken as a basis to the design and implementation of our MAP system. The present chapter summarizes the design of the MAP system, analyzing its structure and its main components.

This chapter is structured as follows: next section presents the MAP system overview, showing the main components of the system and describing their functionality; section 4.2 introduces the language to describe MAP domains and problems; section 4.3 describes the MAP algorithm that planning agents execute in order to obtain a solution to the MAP task, and section 4.4 presents the Partial-Order Planner algorithm and the extensions undertaken to adapt it to a MAP context.

4.1 System overview

Our MAP system follows the theoretical model presented in chapter 3. Hence, it is oriented to a refinement planning approach, in which the refinement plans are obtained through a Partial-Order Planning search process. Figure 4.1 shows the organization of our MAP system, whose main components are listed as follows:

1. **Input files:** Prior to the actual MAP process, the planning agents receive a set of description files, which model the information on the MAP task. Since MAP introduces new requirements that classical planning description languages cannot support, we have designed a MAP language that deals with them.

2. **Partial-Order Planner:** The Partial-Order Planner (POP) is the key element of our refinement planning approach, because there is a POP embedded in each agent of the system. Once a planning agent receives the current base plan, it uses the POP component to generate several refinement plans, which are candidates to be chosen as the next base plan. Each planning agent is provided with this component at the start of the process in order to enable its planning capabilities.

3. **Multi-Agent System:** This component of the system allows the planning agents to communicate and collaborate with others, exchanging partial solutions and selecting the next base plan in order to build a joint solution plan.

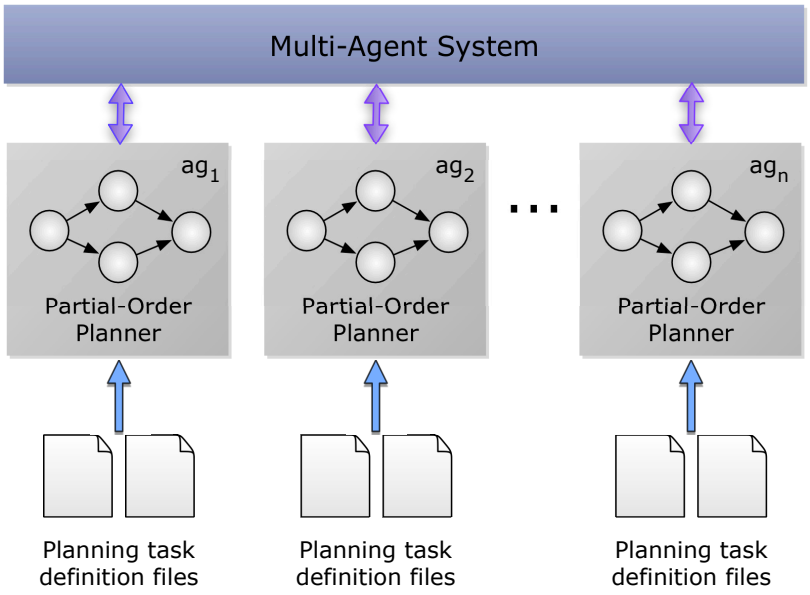

Figure 4.1: Overview of the MAP system design

Planning agents start by receiving and processing the planning task definition files, which configure their internal POP systems. Then, they carry out the MAP algorithm, which consists of two different stages:

- An initial information exchange, which is carried out once, before initiating the planning process. In this stage, agents build a distributed relaxed planning graph (dis-RPG), which will be useful to the MAP process.

- The actual MAP process, by which they exchange and refine partial solutions until reaching a joint solution plan. This stage interleaves two different processes, a coordination phase that allows agents to exchange refinement proposals and select the next base plan, and an individual POP process by which the agents build new refinements over the current base plan.

4.2 Planning language

An important requirement to reach agreements in Multi-Agent Systems is to provide agents with planning capabilities that allow them taking the most appropriate course of action (plan) in order to achieve a specific set of goals. To acquire such capabilities, a planning specification language is required to represent the different elements of a planning task.

Traditionally, planning has been regarded as a single-agent problem, where only one centralized planning entity is required. Hence, MAP presents new requirements and challenges that were not present in classical, centralized planning:

- **Shareable information:** In our approach, planning agents may want to withhold their private information, and decide which information will be shared with the rest of agents. Our planning language includes a `shared-data` section in order to specify which literals will be shared with other agents.

- **Global and private goals:** As opposite to single-agent planning tasks, agents can have individual objectives besides the system's common goals. Hence, our language introduces the `global-goal` and `private-goal` sections to define this information.

As described in section 4.1, all the planning agents in our MAP system receive a set of input files which describe the agent's planning task. Information on a planning task is divided into a *domain* and a *problem* file. The domain file represents the planning actions, the types of objects and the state variables of the task, while the problem file details the actual objects of the task, the initial state i.e. the initial values of the state variables and the task's goals.

As outlined in section 2.4, planning languages have experienced a continuous evolution over the last years, increasing their expressivity continuously by the addition of new features. We have chosen *PDDL3.1* [Kov11], the

most recent version of *PDDL* [GHK+98], as the base of our planning language. However, we have introduced a set of new constructs to the language in order to deal with the additional challenges brought by MAP.

Consequently, this section describes our MAP language approach, specifying in detail its main features. In particular, we analyze the constructs that cover the particular requirements of MAP domains, i.e. modeling the data shared among agents, and the local and global goals. Finally, we present a brief example that shows the modeling of a simple planning task through our MAP language.

4.2.1 General structure

As in *PDDL3.1* classical tasks, each agent on a MAP system has two associated description files that model the agent's planning task. Each domain and problem file has a similar structure to their *PDDL* counterparts, although a few new constructs are introduced, making these files slightly different.

The domain and problem files model the agent's information as a classical *PDDL* domain, defining the objects, predicates, fluents and operators. However, the MAP problems present some distinctive features that have to be reflected in the model.

Firstly, the model should include the information that is shared among the different agents. The `shared-data` construct, defined in the problem file, indicates which object fluents are shared by each agent and with whom. Secondly, unlike classical planning entities, agents in MAP have to fulfill global and local goals. Global goals are the common objectives that a set of agents have to achieve, while local goals refer to an agent's private objectives. Hence, planning agents will propose plans that solve the common goals and satisfy at the same time their own, private interests. The constructs `global-goal` and `private-goal`, defined in the problem files, establish the agent's local and global goals. Finally, a `multi-functions` construct has been included in order to simplify the specification of object fluents in the initial state of an agent.

4.2.2 Shared data

The `shared-data` section, located on the agent's problem file, establishes which data can be shared and with which agent or agents. The information to be shared is defined through object fluents or predicates. The defined construct has the following BNF syntax:

```
<shared-data-def>  ::= (:shared-data <share-def>+)
<share-def>        ::= (<atom-formula-def>+ [- <agent>])
<agent>            ::= <name>
<atom-formula-def> ::= (<predicate> <typed-list(element)>)
<atom-formula-def> ::= (= <object-fluent> <object>)
<predicate>        ::= <name>
<object-fluent>    ::= (<name> <object>*)
<object>           ::= <name>
<element>          ::= <variable> | <constant>
<variable>         ::= ?<name>
<constant>         ::= <name>
<typed-list(x)>    ::= x*
```

As the BNF syntax shows, it is possible to define sets of predicates within the section and associate them to one, some or all the agents in the system (if `agent` is not specified, the predicates are shared with all the agents). Again, the predicates act as patterns, since we consider literals to be the only information shared among agents.

4.2.3 Private and global goals

One of the particularities of MAP when compared to traditional planning is the fact that agents have private and global goals. To reflect this information in the model, the constructs `private-goal` and `global-goal` have been included into the problem files. Similarly to the `goal` section in *PDDL3.1*, the goals can be modeled through predicates or object fluents. The defined constructs use the following BNF syntax:

```
<private-goal-def>  ::= (:private-goal <predicates-def>)
<global-goal-def>   ::= (:global-goal <predicates-def>)
<predicates-def>    ::= <atom-formula-def>
<predicates-def>    ::= (and <atom-form-def> <atom-form-def>+)
<predicates-def>    ::= (or <atom-form-def> <atom-form-def>+)
<atom-form-def>     ::= (<predicate> <typed-list(element)>)
<atom-form-def>     ::= (= <object-fluent-def> <object>)
<predicate>         ::= <name>
<object-fluent-def> ::= (<name> <object>*)
<object>            ::= <name>
<element>           ::= <variable> | <constant>
<variable>          ::= ?<name>
<constant>          ::= <name>
<typed-list(x)>     ::= x*
```

As shown in the BNF syntax description, both sets of global and local goals are described with a conjunction or a disjunction of predicates, or a single one.

4.2.4 Multi-functions

As seen in section 3, our framework considers the explicit representation of true and false information. This makes the encoding process difficult, since it is necessary to encode a much larger volume of data in the initial state of each agent than in a classical planning task. To simplify this encoding process, the `multi-functions` construct has been introduced. This new element of the language is used to describe the planning model's static data (information that does not change throughout the planning process, that is, information that is not modified by the actions' effects).

The multi-functions are defined in the same way as the regular fluents, by declaring them in the domain file. The only difference is that these special functions are declared within their own `:multi-functions` section. The BNF syntax of the construct remains as follows:

```
<multi-func-sec-def> ::= (:multi-functions <multi-func-def>)
<multi-func-def>     ::= (<name> <typed-list(typed-v)>) <type>
<typed-v>            ::= <variable> - <type>
<type>               ::= <name>
<type>               ::= (either <name> <name>+)
<variable>           ::= ?<name>
<typed-list(x)>      ::= x*
```

In the `:init` section of the problem file, the multi-functions are used to actually model the initial state data. The specification of a multi-function within the `:init` section uses the following BNF syntax:

```
<multi-func-def> ::= (= <multi-func> <value-def>)
<multi-func>     ::= (<name> <typed-list(inst-v)>*) <inst-v>+
<inst-v>         ::= <name>
<variable>       ::= ?<name>
<typed-list(x)>  ::= x*
```

4.2.5 Language example

In order to illustrate our language specification, this section presents a simple planning task example. The example models a transport example, in which agents act as transport agencies that use their trucks to deliver their packages to a certain city. Agents work in a certain geographic area, which means that they can only deliver packages in cities within their work area. To deliver packages outside their area successfully, they will have to work jointly with the rest of agents in the system.

Agents know accurately the situation of their trucks and the position of the packages they have to collect and deliver. They also know the different

geographical areas, the situation of the cities within their work area, and the distance between each pair of cities connected by a road. However, they do not have a precise knowledge about the geographical situation of the cities outside their influence area, since it is not relevant for their planning models. In the following, we present the modeling of this planning task, putting emphasis on the encoding of the new constructs introduced in our MAP language.

The present section describes the most relevant information included in the domain and problem files. In this case, the domain file is shared by the three agents present in the system, as they have the same capabilities. The first section of the code specifies the types of objects included in the formalization:

```
(:types truck package agent city - object)
```

The `truck` objects are in charge of carrying the packages to their destinations. Each truck belongs to a certain agent, and can only be employed by its owner. The `Package` objects have to be picked up and delivered by the trucks. `Agents` are also modeled as objects, since each one knows the rest of agents present in the system. Finally, agents know the location of each `city` within their geographical area.

Once the types have been defined, the code specifies the domain's object fluents as follows:

```
(:fluents (at ?t - truck) - city
          (in ?p - package) - (either truck city)
)
```

The `at` fluent indicates that a `truck` is placed on a certain `city`, while the `in` fluent indicates that a `package` is located in a certain `truck` or `city`.

The domain file includes also a multi-function, which is described as follows:

```
(:multi-functions
    (link ?c1 - city) - city
)
```

Finally, the domain file describes the actions that the agents can perform. In this example, agents can carry out three different actions:

```
(:action Load
    :parameters (?p - package ?t - truck ?c - city)
    :precondition (and (= (in ?p) ?c)
                       (= (at ?t) ?c))
    :effect  (assign (in ?p) ?t)
)
(:action Unload
    :parameters (?p - package ?t - truck ?c - city)
    :precondition (and (= (in ?p) ?t)
                       (= (at ?t) ?c))
    :effect (assign (in ?p) ?c)
)
(:durative-action Drive
    :parameters (?t - truck ?c1 ?c2 - city)
    :precondition (and (= (at ?t) ?c1)
                       (member (link ?c1) ?c2))
    :effect (assign (at ?t) ?c2)
)
```

The `Load(?p ?t ?c)` action indicates that the package `?p` is loaded onto the truck `?t`; both package and truck are located at the city `?c`. `Unload(?p ?t ?c)` causes the package `?p` to be unloaded from the truck `?t`; both package and truck are located at the city `?c`. Lastly, `Drive(?t ?c1 ?c2)` indicates that the truck `?t` goes from the city `?c1` to the city `?c2`.

Notice that the object fluents are introduced in the actions' preconditions by using the `=` operator, while the multi-functions use the `member` operator. This operator indicates that the value is part of the multi-function's domain.

Unlike in the case of the domain file, each agent has its own problem file. Each of them includes the problem objects, goals, initial state, shared data and mapping information. The following fragment of code includes the whole problem file defined for agent 1:

```
(define (problem Ag1)
(:domain Transport)
(:objects
    t1 t2 - truck
    Madrid Toledo Cuenca
    Ademuz Valencia Sagunto - city
    p1 p2 - package
)
```

```
(:init
    (= (link Madrid) {Toledo})
    (= (link Toledo) {Madrid Cuenca})
    (= (link Cuenca) {Madrid Ademuz Valencia})
    (= (link Ademuz) {Cuenca Valencia})
    (= (link Valencia) {Cuenca Sagunto Ademuz})
    (= (link Sagunto) {Valencia})
    (not (= (link Madrid) {Cuenca Ademuz Valencia Sagunto}))
    (not (= (link Toledo) {Ademuz Valencia Sagunto}))
    (not (= (link Cuenca) {Toledo Sagunto}))
    (not (= (link Ademuz) {Madrid Toledo Sagunto}))
    (not (= (link Valencia) {Madrid Toledo Cuenca}))
    (not (= (link Sagunto) {Madrid Toledo Cuenca Ademuz}))
    (= (in p1) Toledo) (= (in p2) Cuenca)
    (= (at t1) Madrid) (= (at t2) Sagunto)
)

(:global-goal  (in p1 Valencia))
(:private-goal (in p2 Sagunto))

(:shared-data
      (in ?p - package Ademuz) - ag2
      (at ?t - truck) - ag3
)
)
```

As the code indicates, the problem file defines the situation of the cities and the connections and distances between them, the location of the trucks and packages, the global goals and the local objectives of the agent. The `:shared-data` section defines which information is shared with agents 2 and 3.

4.3 Multi-Agent Planning algorithm

This section details the MAP algorithm followed by the agents to devise, exchange and select partial-order plans, in order to come up with a solution for the MAP problem. In the current version of the MAP system, the coordination of agents is performed through a planning protocol in which one agent leads the process of gathering the new refinements and selecting the next base plan. However, our final goal is to perform coordination through an argumentation process. This argumentation framework is currently at a theoretical stage; its further development and integration with the MAP system constitutes one of our lines of future work (see section 6.2 for detailed information).

Our MAP algorithm is divided in several stages, that can be outlined as follows:

- **Initial information exchange:** The MAP algorithm starts with an initial communication stage by which the agents exchange some information on the planning domain, in order to generate some data structures that will be useful in the subsequent planning process.

- **Problem-solving algorithm:** The actual planning process is carried out at this stage. It comprises two different stages that are interleaved, a Multi-agent System (MAS) coordination process that allows agents to exchange the generated refinement plans and to select the next base plan, and an internal planning process by which the agents refine the current base plan:

  - **Refinement planning process:** Agents try to refine the current base plan of the MAP system individually. In order to perform this task, each planning agent is provided with an internal Partial-Order Planning (POP) system. We have carried out some adaptations to the classical POP algorithm to extend it to a MAP context (see section 4.4).
  - **MAS coordination process:** This stage allows the agents to communicate and exchange the refinements over the current base plan they have individually generated. Once the agents are informed about the available refinements, they choose the following base plan.

The following sections detail the preliminary information exchanging stage performed by the agents and the actual problem-solving algorithm, including the MAS coordination process and the construction of the refinements carried out by the Partial-Order Planner embedded in each planning agent.

4.3.1 Initial information exchange

Prior to the actual MAP process, agents perform a preliminary stage which allows them to share planning information effectively. This initial stage focuses on the construction of a distributed Relaxed Planning Graph (dis-RPG), based on the approach in [ZNK07]. The dis-RPG provides the agents with valuable planning information that will be used throughout the problem-solving process:

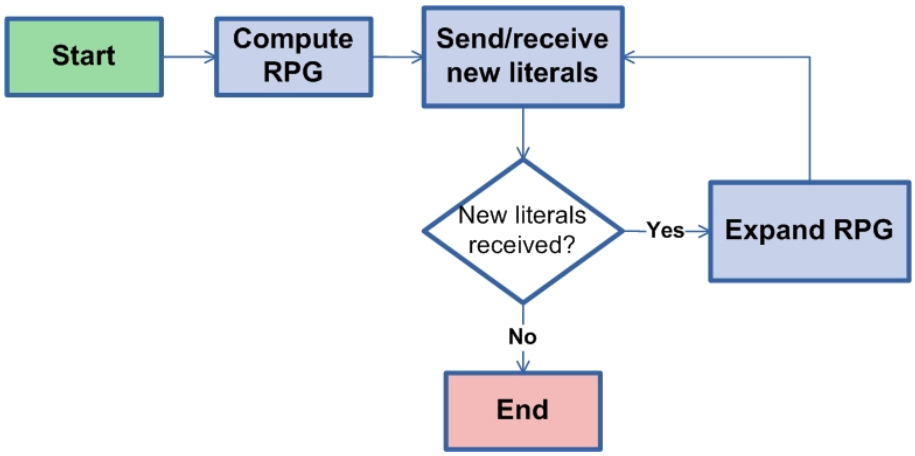

Figure 4.2: Dis-RPG construction algorithm

- It allows agents to effectively exchange the literals defined as shareable in the MAP domain definition files, through the `:shared-data` construct (see section 4.2.2). After this stage, each literal is labeled with a list of agents that can achieve it (and have shared this information), which gives agents a more accurate view of the planning domain. As section 4.3.2 outlines, this information will be useful to compute refinement plans.

- It computes an estimate of the best cost to achieve each literal, a handy information to devise heuristics to guide the problem-solving process.

Despite this process performs the distributed construction of a planning graph, none of the agents handle a complete representation of the dis-RPG. In contrast, each agent maintains its own Relaxed Planning Graph (RPG) internally, keeping this way the private data occluded to the rest of participant agents.

Figure 4.2 depicts the dis-RPG building process. Firstly, each agent computes an initial RPG taking into account only its own actions and literals. For the construction of this initial planning graph we follow the algorithm presented in [HN01]. Once each agent has computed an initial RPG, the dis-RPG composition begins. This construction process consists of the following stages:

- **Literals exchange**. Agents share the literals included in their RPGs, according to the `shared-data` section of their planning domain definition files. This way, if a literal in the agent's RPG matches with

a predicate in the `shared-data` section, it will be sent to the agents specified in this section.

- **RPG expansion**. After sending and receiving the new literals, each agent updates its RPG as follows:
  - If a literal is not yet in the agent's RPG, it is stored according to the literal's cost.
  - If the literal is already in the RPG, the agent checks the cost of the received literal; if this cost is lower than the cost the agent has registered in the RPG, then the agent updates the cost. This way, the agent stores only the best estimated cost to reach each literal.

  Once the RPG is updated the agent performs an expansion process, checking if the new literals trigger the appearance of new actions in the RPG. The effects of these new actions will be shared in the following literals exchange stage.

The process finishes when there are no new literals in the system. After exchanging planning data, agents start the MAP process to devise a solution plan jointly.

4.3.2 Problem-solving algorithm

After the initial information exchange, agents initiate the problem-solving protocol (see figure 4.3). The algorithm comprises two interleaved stages: the refinement planning stage and the coordination stage. Once completed the interaction of one stage, agents carry out an interaction of the other stage. On the one hand, agents build individual refinements over the current base plan by using a Partial-Order Planner. On the one hand, agents follow a coordination process to gradually build a joint solution plan for the MAP task, exchanging and evaluating the refinements generated individually and selecting the most promising one in order to reach a solution.

4.3.2.1 Refinement planning via POP

Each agent in the system executes an individual Partial-Order Planning (POP) process in order to refine the current base plan. The current base plan is taken by each agent's planner as the initial plan, and it obtains a set of valid refinements of the current base plan.

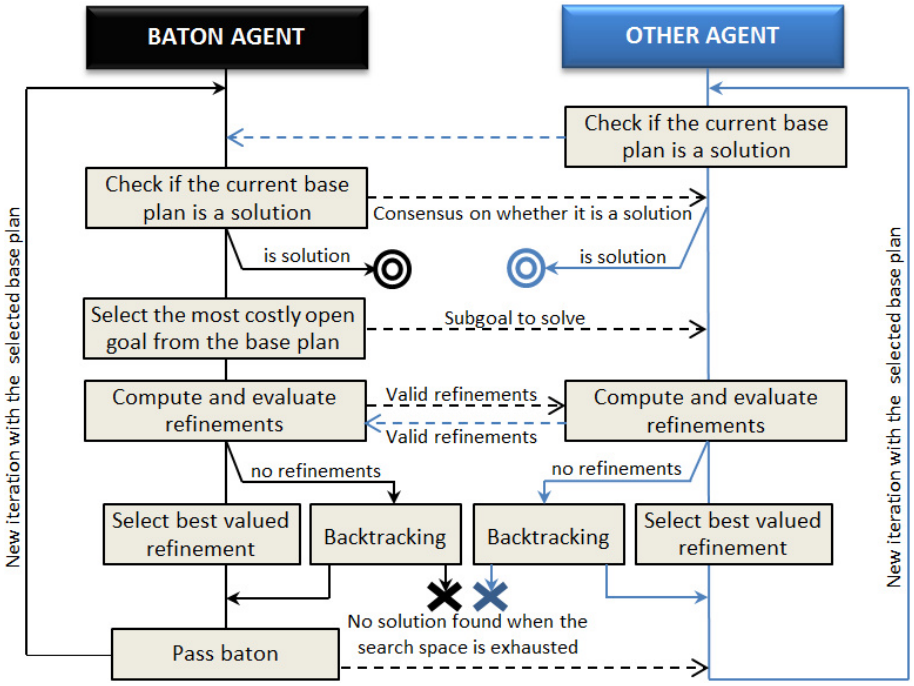

Figure 4.3: Sequence diagram of the problem-solving algorithm

The POP process applies a heuristic function in order to guide the search. Currently, we use a heuristic function (the *SUM heuristic*) based on the sum of the costs of the open goals in the Relaxed Planning Graph (see section 4.4.3 for more information on POP heuristics). This heuristic is also used to evaluate the refinement plans in the coordination stage.

The planning process solves the selected subgoal, and all the subgoals that arise from this initial resolution in a cascading fashion, leaving the rest of preconditions unsolved. This way, the valid refinements obtained by the agents in the POP process are evaluated in the next step of the coordination process, in order to select the next base plan among them.

We have designed a customized version of the classical POP algorithm in order to satisfy the requirements introduced by the MAP approach. Section 4.4 discusses the POP design and describes in detail the most relevant changes brought into the POP algorithm in order to cope with MAP domains.

4.3.2.2 Multi-Agent System coordination protocol

The coordination protocol is based on a democratic leadership, in which a leadership baton is scheduled among the agents following a round-robin strategy. The algorithm undertakes several iterations of the coordination stage,

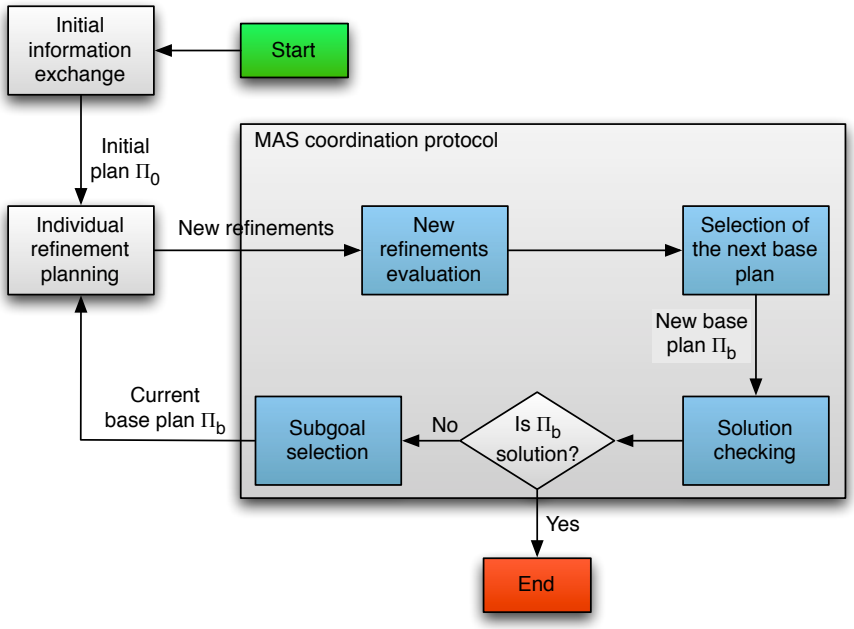

Figure 4.4: Stages of problem-solving algorithm

leaded by the agent which has the baton (baton agent). Once a coordination stage is completed, the baton is handed over to the following agent. A coordination stage starts once the agents have performed the individual refinement planning stage through their embedded POP systems (see figure 4.4) and consists of the following steps:

1. **New refinements evaluation:** In this step, agents evaluate the partial plans obtained in the refinement planning stage (see section 4.3.2.1). The obtained refinements are communicated to the rest of agents for their evaluation. This evaluation is performed through a POP heuristic function that estimates the quality of a refinement, that is, to what extent it can be refined to a solution plan (the lower value, the better quality). The quality of this function directly affects the planning efficiency. Although we are currently using a heuristic function, we aim to enhance the whole refinement evaluation mechanism in the future through the introduction of an argumentative process (see section 6.2), which will provide a social approach to the evaluation and selection of the refinements. As outlined in section 4.3.2.1, we are currently using a heuristic function based on the sum of the costs of the open goals in the Relaxed Planning Graph (the *SUM heuristic*). This POP heuristic

function is also used to guide the internal POP process. Section 4.4.3 details the different heuristics we have considered for our MAP model.

2. **Selection of the next base plan**. Agents select the best valued refinement over all the refinement plans generated until this point as the new base plan Π_b.

3. **Solution checking:** In this step, agents decide whether the current base plan Π_b is a solution plan or not. Taking into account that base plans are threat free and that the problem goals are preconditions of the final fictitious action, a base plan is a solution if it does not have open goals. Since some open goals might not be visible to some agents, all agents must confirm there are no unsupported preconditions in the plan. The baton agent is in charge of collecting the other agents' opinion and sending back the reached consensus.

4. **Subgoal selection:** The baton agent selects the following open goal sg to be solved ($sg \in openGoals(\Pi_b)$) and communicates its decision to the other agents. The subgoal is selected as the most costly open goal, according to the dis-RPG the baton agent can observe. This is a simple selection mechanism but, in general, produces good results. It is important to notice that the baton agent may not see any of the remaining open goals of the base plan; in this case, it will pass the baton to the following agent to complete this step. The selection of the subgoal to be solved starts a new refinement planning stage, by which the agents will refine the current base plan Π_b.

Hence, the coordination algorithm can be seen as an A* search process [RN03], in which the refinement plans are the nodes in the search tree and each planning stage expands a node in the search tree.

4.4 Partial-Order Planner

As shown in section 4.1, the core element of our MAP system is the planning module, which provides every agent in the system with planning capabilities. The designed component is based on the Partial-Order Planning (POP) paradigm, since this framework offers some interesting features for our refinement planning approach. This section describes briefly the classical POP approach and the adaptations performed to the classical POP algorithm to extend it to a MAP context.

4.4.1 Classical Partial-Order Planning algorithm

In Total-Order Planning approaches, the plan's actions are obtained in the same order in which they are executed. This way, if a planner chooses a wrong action, it will have to introduce another action to undo the effects of the first one. As opposite to these models, the Partial-Order Planning (POP) paradigm [BW94] introduces a more flexible approach, establishing partial-order relations between the actions in the plan rather than enforcing a strong, concrete order among them. POP-based planners 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 it. This strategy based on deferring decisions during the planning search is known as *least commitment* [Wel94].

Instead of performing a state space search, POP models adopt a plan space search approach. Hence, a POP addresses the process of building a search tree in which each node represents a different partial plan, without maintaining the notion of planning state. POP is also classified into the backward-chaining search approaches, since it begins the search by satisfying the problem goals, and builds the plan backwards. In conclusion, POP can be considered a plan-based, backwards chaining search process.

This section presents the classical POP algorithm, using the nomenclature for the main components of a partial-order plan introduced in section 3.2.

Algorithm 1 Classical POP algorithm

$Open\_nodes \leftarrow$ {Void plan}
repeat
  Select $\Pi \in Open\_nodes$
  $Pending\_flaws \leftarrow open\_goals(\Pi) \cup threats(\Pi)$
  if $Pending\_flaws = \emptyset$ **then**
    return Π
  end if
  Select and extract $\Phi \in Pending\_flaws$
  $Successors \leftarrow \{\Pi_r\}$, $\forall\ \Pi_r$ that solves Φ
  if $Successors \neq \emptyset$ **then**
    $Open\_nodes \leftarrow Open\_nodes \cup Successors$
  end if
until $Open\_nodes = \emptyset$
return fail

Algorithm 1 outlines the classical POP process. The classical algorithm explores a search tree in which each node represents a partial plan. The search

process starts with an initial void plan, which only includes the fictitious steps $\mathcal{I}$ and $\mathcal{F}$. This plan is introduced on a *Open_nodes* list, which will store the leaf nodes of the search tree.

At each iteration, the search algorithm extracts a plan from *Open_nodes* and selects and solves one of its pending flaws (an open goal or a threat). This points out the two key decision points of the POP algorithm: plan selection and flaw selection. These decision points will determine the performance of the search process, as they will direct the expansion of the search tree.

Once the plan's flaw is selected, the search process will solve it. The resolution of the flaw entails the generation of a set of refinement plans that constitute the successors of the current plan. The POP system will create as many refinement plans as different ways to solve the flaw. As explained previously, an open goal is addressed with the introduction of a new causal link in the plan, using an existing step or inserting a new one, while a flaw is solved by promoting or demoting the threatening step. The current plan's successors will be introduced in the *Open_nodes* as candidates to be refined in the subsequent iterations of the process.

Once a plan which has no flaws is selected, it will be returned as a solution and the planning process will end successfully. If the *Open_nodes* list gets empty before finding a solution, the planner process will end with a failure. The planner considers that there is no solution for the formulated planning domain, since it has explored the whole search space without finding a solution plan.

4.4.2 POP extensions for refinement planning

Our MAP approach, based on refinement planning, introduces new requirements that make it necessary to introduce several changes to the classical POP algorithm. We have implemented a customized version of the classical POP algorithm, in order to satisfy these new requirements. The most relevant changes brought into the algorithm can be summarized as follows:

1. **Initial plan:** The classical POP algorithm starts with a void plan, which contains only the fictitious initial and final steps. Our implementation allows to choose any partial plan as the initial plan. This way, the agents are able to establish the current base plan as the initial plan of the POP process, in order to obtain the consequent refinements.

2. **Subgoal resolution**. Our POP implementation solves only the open goal selected by the agents as the current subgoal. The rest of open goals in the initial plan are ignored. Once the current subgoal is supported by a causal link, the planner will try to solve, in a cascading

fashion, the new open goals resulting from the resolution of the current subgoal and are only solvable by the agent which is performing the planning process (the dis-RPG provides the information on which agents can solve a open goal).

3. **Solution checking**. Classical POPs return only solution plans, i.e. threat-free plans without open goals. Since our planner must generate refinement plans, not necessarily solutions, we have redefined the solution checking stage of the algorithm. In order to be validated by the POP as a refinement plan, a partial-order plan must meet the following conditions:

   - It must be threat and mutex free.
   - It must have a causal link that supports the current subgoal.
   - If the plan has added new open goals over the current base plan, each one of them must be solvable by another agent or group of agents. This way, the planning agent will only support a new open goal if it is the only agent in the system able to solve it (the dis-RPG provides information on which agents can solve an open goal).

4. **Planning restarting**. Typically, the POP process finishes once a solution is found. Nevertheless, our POP allows to resume the planning process at will if the agent wants to obtain more refinements to the base plan.

Consequently, our POP implementation is aimed at provide the planning agents with refinement proposals that will be analyzed in the MAP process in order to find a solution for the planning task.

4.4.3 POP heuristic functions

Our POP system applies an informed search approach i.e. it selects the next partial plan to be refined by using a heuristic function [RN03]. A heuristic is an evaluation function that returns a numeric value that denotes the desirability of refining a certain partial plan. The heuristic value for a given refinement plan Π is expressed as $f(\Pi) = g(\Pi) + h(\Pi)$, where $g(\Pi)$ represents the cost of reaching the current partial plan from the initial one, and $h(\Pi)$ estimates the cost of reaching a solution plan from the current refinement.

The efficiency of the POP search process is strongly related to the heuristic function adopted. However, as outlined in section 2.3, one of the main

shortcomings of the Partial-Order Planning (POP) paradigm is the absence of a competitive search heuristic. The most recent efforts in this direction, based on estimating the cost of the plans' open goals, present poor results that cannot compare with the performance of the most recent state-based approaches.

Therefore, we have taken into consideration some of the most recent attempts to define a high-quality heuristic function for POP [NK01], as well as devising our own evaluation function.

The following sections summarize the different heuristics we have considered for our MAP framework. More precisely, we have adopted two of the state-of-the-art POP heuristics (SUM and MAX heuristics), which estimate the plan's quality by studying the cost of its open goals in the Relaxed Planning Graph. We also outline the basics of our in-development heuristic function (the DTG heuristic), which analyzes the value transitions of the state variables to estimate a solution plan.

Section 5.3.2.1 compares the performance of the SUM and MAX heuristics. Due to its experimental nature, the DTG heuristic function has been excluded from the comparison, since it requires further improvements to be competitive against the other heuristics (see section 6.2 for details).

4.4.3.1 SUM heuristic

This heuristic estimates the cost of reaching a solution from a certain partial plan by calculating the sum of the costs of the plan's open goals. Although it is not an admissible heuristic (it may overestimate the cost of reaching a solution), the SUM heuristic has shown a good performance on most of the tests (see section 5.3). The SUM heuristic function, $f(\Pi) = g(\Pi) + h(\Pi)$, is computed as follows:

- $g(\Pi)$ is computed as the number of actions (except for the fictitious ones) of the current refinement Π.
- $h(\Pi)$ is defined as the sum of the costs of the refinement's open goals $\mathcal{OC}(\Pi)$ in the Relaxed Planning Graph.

4.4.3.2 MAX heuristic

The MAX heuristic is similar to the SUM heuristic, as it evaluates a partial plan by analyzing the costs of the plan's open goals. However, it tries to minimize overestimation by computing the cost of reaching a solution as the cost of the most costly open condition in the partial plan. Hence, the MAX heuristic, $f(\Pi) = g(\Pi) + h(\Pi)$, is computed as follows:

- $g(\Pi)$ is computed as the number of actions (except for the fictitious ones) of the current refinement Π.
- $h(\Pi)$ is defined as the cost in the Relaxed Planning Graph of the most costly of the open goals $\mathcal{OC}(\Pi)$.

4.4.3.3 DTG heuristic

The heuristic functions presented in the previous sections, based on estimating the cost of the plans' open goals, present poor results that cannot compare with the performance of the most recent state-based approaches. Therefore, one of the challenges we face to improve the performance of our MAP system is the design of a reliable heuristic that guides the POP search process efficiently.

The introduction of the *PDDL3.1* language, and more precisely, the use of state variables to model the literals of the planning task provides new and valuable information that was not available in propositional approaches. It is possible to take advantage of this information to devise new and more powerful search heuristics for POP.

The work in [Hel04] introduces the *Domain Transition Graphs* (DTGs). The procedure described in this paper analyzes the planning task's actions and generates, for each state variable, a directed graph that represents the ways in which the variable can change its value. Each transition on a DTG is also labeled with the necessary conditions for the transition to occur i.e. those preconditions that figure on all the planning actions that perform the value transition.

Our heuristic relies on the DTGs for building an intermediate data structure that estimates the layout of a solution plan for the planning task, the Plan Transition Graph (PTG). Our purpose is to use this data structure to guide the search process efficiently. To do so, we have designed a heuristic that maps the plan's actions to the PTG's ones, and estimates the quality of a partial plan according to its similarity to the PTG.

However, our heuristic is still in development, and it is not yet competitive against the SUM and MAX heuristics. Section 6.2 summarizes the research we will carry out in the following months in order to improve the DTG heuristic.

Chapter 5

Multi-Agent Planning system implementation

As well as designing the components of the MAP system, we have carried out an initial implementation of the system has. The developed software constitutes an essential framework that will be used as a base for the development of our future research lines.

The present chapter analyzes the implementation details of the main components of the MAP system, outlining the structure and features of each component, along with the technologies used to develop them. Finally, we present the experimental results obtained after testing the system with a set of planning tasks. These results evaluate the quality of the current implementation of our MAP system.

5.1 Multi-Agent System implementation details

Our Multi-Agent Planning system has been developed under the Magentix2 platform [FAS+10]. Magentix2 has been chosen since it is aimed at simplifying the development of multi-agent systems and ensuring standard compliance with the FIPA specifications [ON98] through a comprehensive set of system services: naming and yellow-page service (DF), message transport and parsing service (MTS), and a library of FIPA interaction protocols ready to be used.

A planning agent extends a Magentix2 single agent with (distributed) planning capabilities. This means that a planning agent is able to:

- Compute a (partial-order) plan to achieve a set of propositional goals, as each planning agent has a Partial-Order Planner (POP) at its dis-

posal. This way, planning agents can be used as planning services for the purposes of other agents.

- Collaborate with other planning agents to build a joint plan that reach the goals. This feature allows solving planning tasks which are distributed among several agents.

Figure 5.1 shows the integration of the planning agents in a Magentix2 platform. When a planning agent enters in the system, it is registered in the DF (Directory Facilitator) and a planning service is associated to it. The planning tasks are encoded in several *PDDL3.1* input files. When a task requires the collaboration of other planning agents, they are searched in the DF and the joint plan building process begins.

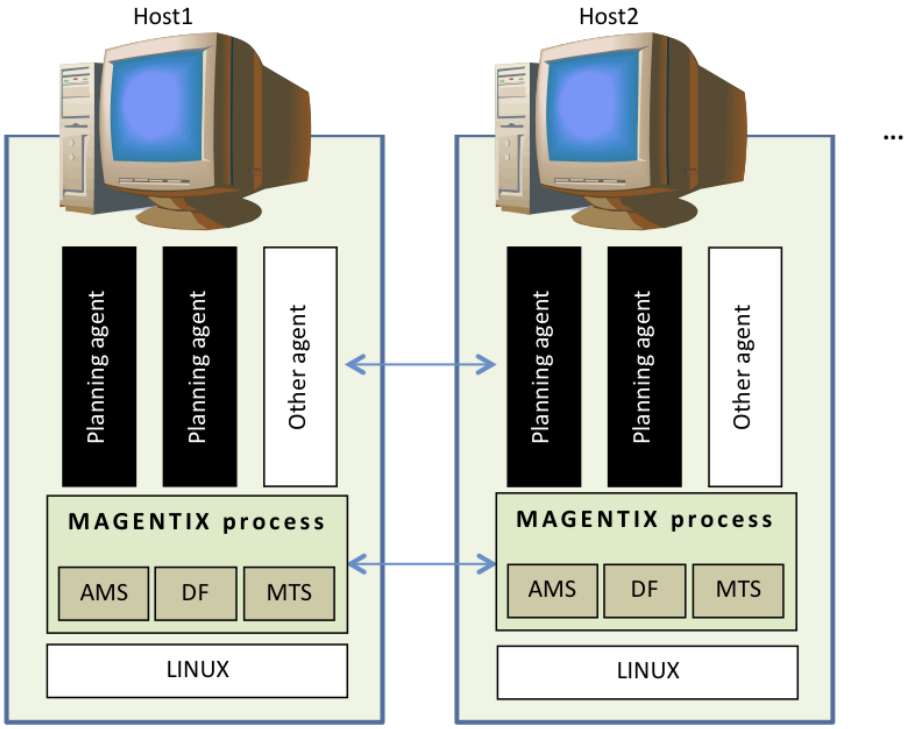

Figure 5.1: Structure of the Magentix platform

The AMS and DF services are implemented as Linux processes, children of the Magentix2 process, which are replicated in each computer in the platform.

In turn, Magentix2 agents, including the planning agents, are Linux processes, children of the AMS process. This way, the AMS process has the same control that a Linux process has over its children, which includes creation, termination, completion status information, etc. With this design we have a

tree of processes in each computer whose root is the Magentix2 process and its leaves are the agents being executed in that computer.

Planning agents wait in an idle state for a planning request. When it is received, the agent analyzes whether the planning task must be solved collaboratively or not; this piece of information is explicitly defined in the PDDL-based description files of the task. If the agent can solve the task on its own, it uses the internal POP to find a solution plan; in this case, our planner behaves like a classical single-agent Partial-Order Planner. This behaviour can be observed in Figure 5.2.

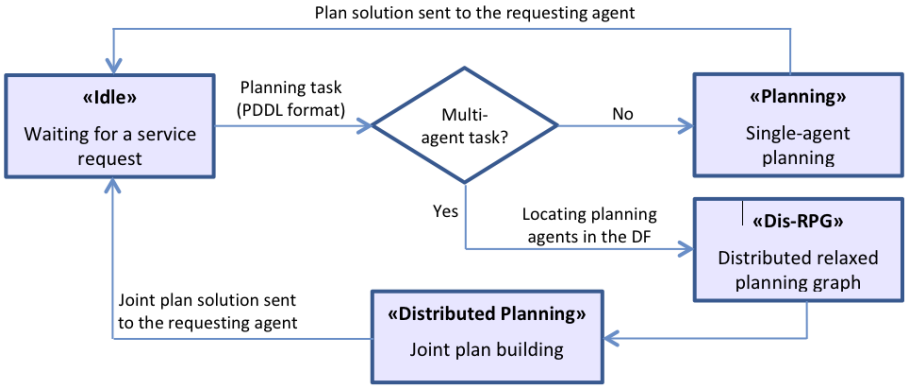

Figure 5.2: State diagram of a planning agent

When the task to solve is distributed among several planning agents, they collaborate to find a joint solution plan. As described in section 4.3, the plan building process has two main stages:

1. First, a distributed relaxed planning graph (dis-RPG) is built. Agents inform the other ones about the propositions they likely can achieve (this data is heuristically calculated). Agents, of course, only communicate their public information during this process. As a result, the dis-RPG construction is very fast and provides the agents with useful information for the next stage.

2. In the second stage, agents build the joint plan collaboratively. They use their internal POP to compute refinements for the existing partial plans, thus incrementally building the joint solution plan.

5.2 Partial-Order Planner implementation details

The Partial-Order Planner component represents the core element of our MAP system, and it is also in the center of some of the future research lines. Hence, we have carried out a modular implementation that allows this component to be flexible, reusable and easily extensible.

The POP software can be used not only as part of the MAP system, but also as a standalone planner. Furthermore, it can be integrated in a service-oriented platform, and its modules can be reused independently if necessary. This section outlines the structure of the implemented POP system, its main features and the technologies used in its development.

5.2.1 POP implementation structure

Our Partial-Order Planner (POP) implementation, fully based on the Java language [Gos00], is structured in three different modules: the instancing module, the grounding module and the planning module. Each module provides a well-defined interface, which allow them to be reused independently if needed. The functionality provided by each module is summarized as follows:

1. **Instancing module:** This module takes the agent's domain and problem description files and processes the information into Java classes. The input files are written in our *PDDL3.1*-based MAP domain definition language described in section 4.2. The parser module is also ready to process regular *PDDL3.1* domains.

2. **Grounding module:** Our POP implementation works with fully instanced planning tasks. This requirement makes necessary an intermediate module that instances the tsk's actions and fluents. The grounding module solves all the variables in the actions and fluents, generating all the possible instanced actions and fluents. The grounding algorithm performs a reachability analysis, which discards those instanced actions and fluents that are not reachable from the initial state. Finally, the grounding module is also in charge of generating the dis-RPG described in section 4.3.

3. **Planning module:** This module implements the actual POP search process. As explained in the previous sections, our POP implementation allows to take any partial plan as the initial plan. It is also able to restart the search after generating a valid refinement. The planner can also work with regular, single-agent planning domains.

The POP modules have also been implemented to be used in a service-oriented platform. All the modules are implemented as bundles that follow the OSGi standard [All03]. When integrated in an OSGi service-oriented platform, the system will be able to provide the agents with a fully configured POP, which they will be able to handle through the POP interface.

5.2.2 POP main features

Our POP implementation has been devised to be used as the core of the MAP system or as a standalone planner. The implementation is also extensible and customizable, allowing the user to develop new search methods and heuristics, among other customizable elements. The most relevant features of our POP implementation are listed as follows:

1. **Modularity:** As explained in the previous section, the modules that compose the Partial-Order Planner, and be used independently, and as they have been designed as OSGi bundles, they can be easily integrated in a service-oriented system.

2. **Flexibility:** Our POP can be used both with MAP domains (as a part of our MAP system) or with regular planning domain (as a standalone application). The implementation implementation is strongly based on Java interfaces, which allow the user to modify and customize certain aspects of the planning process, the more relevant of which are the following ones:

   - **Search strategy:** The search strategy is a key aspect of the planning process, since it controls the exploration of the plan space. The current implementation of the POP supports several uninformed search strategies, such as depth-first or breadth-first search methods. It also supports several heuristic (informed) search strategies, such as A* or Iterative Deepening A* [RN03]. The user is allowed to implement other search strategies, whether they are uninformed or informed.
   - **Plan selection heuristic:** One of the key points of the POP algorithm is the selection of the next plan to be refined. Our implementation provides some of the most popular POP heuristics [NK01] as well as a set of custom heuristics, based on the notion of the dis-Graph introduced in section 4.3.1. Again, the user can define new heuristics and integrate them with the POP.

- **Flaw selection strategy:** The second key decision point of the POP search algorithm comes up when selecting the next flaw to solve. There are several flaw selection strategies already defined for POP [PJP97], concerning the moment when the plan's threats get solved (its resolution can be deferred till the plan has no unsolved conditions, or performed immediately after being detected), and the selection of an unsolved condition. Currently, our POP implementation gives priority to the threat resolution, and uses a custom multi-criteria to cope with the open goals (we choose firstly the precondition that generates less successors, and apply other criteria in case of a tie). However, the user may redefine the flaw selection criteria if needed.

- **Solution checking:** As seen in section 4.4.2, one of the modifications performed to the classical POP algorithm to adapt it to a MAP context deals with the solution checking. Hence, our POP implementation can return solution plans when working as a regular planner or valid refinements, in a MAP context. This part of the algorithm is also fully customizable, which allows us to modify it in case we change the notion of solution plan.

3. **Performance:** The POP implementation has been carefully designed in order to maintain a compromise between time and space cost. To do so, plans are stored in an incremental fashion. Each new node added to the search tree contains only the new information generated over its parent node (the partial plan it has refined). Data structures have also been designed to be reused over the planning process as much as possible. This reduces dramatically the memory consumption, which is an important aspect in planning, due to the massive dimensions the search trees can get in medium-sized problems. The implementation has also been tweaked in order to take profit of its internal structure and speed up the code. This way, the implementation keeps a sensible equilibrium between memory and time consumption.

Hence, the POP implementation offers a flexible and modular approach, while keeping an adequate performance. The flexibility is an important requirement being our MAP system a work in progress. Section 6.2 introduces the next extensions we are about to perform to the MAP system, and particularly to the POP.

5.3 Experimental results

The MAP system implementation has been put to test through the execution of several tests. The tests compare the performance and the quality of the solution plans between MAP tasks and their centralized counterparts. We have also performed several tests to analyze the performance of our POP heuristic against other state-of-the-art POP heuristics.

These tests performed involve a set of planning tasks, defined over three different planning domains.We have defined several multi-agent and single-agent problems for each of these planning domains. Next sections present the planning domains defined and analyze the results of the different tests.

5.3.1 MAP domains

The planning domains devised to try the MAP system have been taken from real-life problems or adapted from well-known case studies. They are designed to ease the definition of distributed problems, but they also allow the modeling of centralized problems.

5.3.1.1 Transport domain

The first planning domain we will use to put our MAP system to work is depicted in Figure 5.3. This domain presents a a transportation and a storage scenario, in which agents can carry out the role of a driver or manage a storage facility. The objective of the driver agents is to transport a set of packages between the warehouse and a set of cities. In turn, warehouse agents are in charge of storing packages or loading the trucks.

Truck agents are in charge of transporting packages through a network of cities. The domain specifies bidirectional links among cities, that can be used by the trucks to move from one city to another. Trucks can only travel within the cities included in their working areas. This way, they have to work together with other agents in order to send packages to a different area. Truck agents have the following capabilities:

- `load`: Loads a `package` into a `truck`. Both the `package` and the `truck` must be located at a `city` within the `agent`'s `area`.
- `unload`: Unloads a `package` from a `truck` located at a `city` within the `agent`'s `area`.
- `drive`: Drives the `truck` from a `city` to another one. Both cities must be within the `agent`'s `area`.

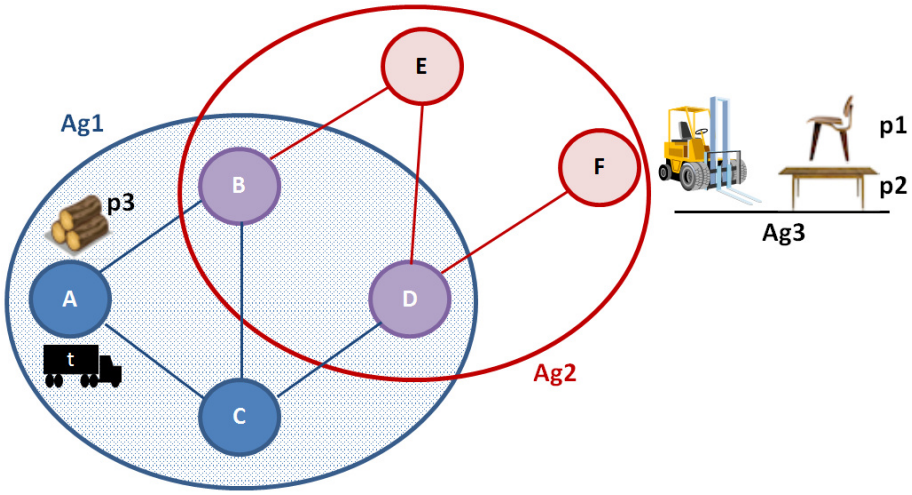

Figure 5.3: Transport domain example

The warehouse domain is similar to the classical *blocksworld* domain, in which packages can be stacked and unstacked on/from the table or other packages. In this case, however, there is room in the table for just one stack of packages, and there are two types of packages, raw materials and final products. The warehouse agent can deliver final products to the city adjacent to the warehouse (the exchange city), and acquire raw materials previously unloaded by the trucks in the exchange city. The warehouse agents can perform the following actions:

- `acquire`: Obtains a `raw-material` from the `exchange-city`.
- `deliver`: Delivers a `final-product` to the `exchange-city`.
- `stack`: Stacks a `package` over another `package`, or puts it down on the `table`, in case it is empty.
- `unstack`: Unstacks a a `package` from another `package`, or picks it up from the `table`, in case it is empty.

The centralized problems based on this domain are defined by introducing the `carrier` and `depot` types, which represent the truck and warehouse agents. This allows to formulate single-agent problems, in which the centralized planning entity devises a plan for all the workers in the system.

5.3.1.2 Picture domain

This domain, adapted from the case study presented in [PSJ98], presents a situation in which several workers have to work together to hang a set of

pictures on walls. To do so, they can acquire different tools that are scattered over several rooms. Hence, agents must move through the rooms to obtain the tools and hang the pictures. The domain establishes a set of bidirectional links that indicate the connections among the different locations.

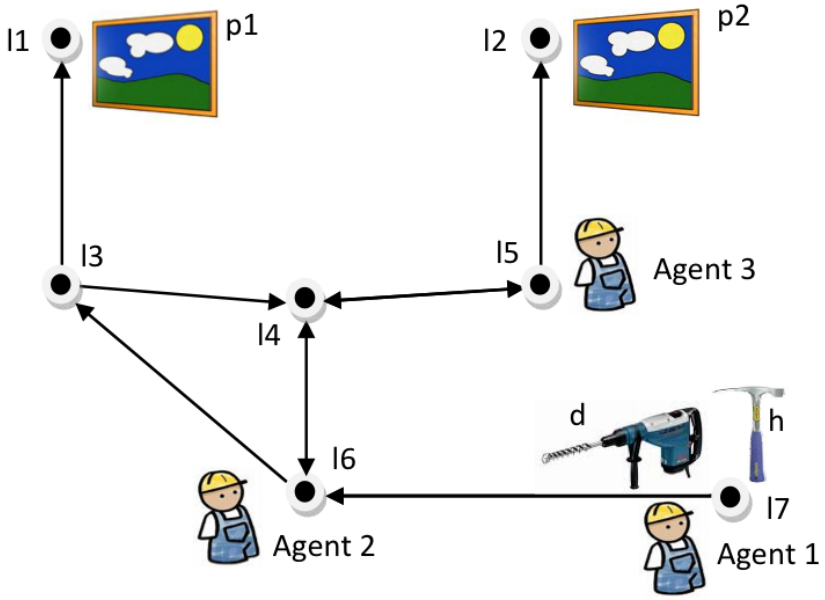

Figure 5.4: Picture domain example

Figure 5.4 depicts an example of this planning domain. In contrast to the transport domain, agents in the picture domain are not specialized, they share the same capabilities, which are listed as follows:

- `pickUp`: Obtains a `tool` from a `location`. The `agent` and the `tool` must be situated at the same `location`.
- `putDown`: Puts down a `tool` in a certain `location`. The `agent` must be carrying the `tool` in order to put the tool down.
- `pass`: Passes the tool to another `agent`. Both agents must be at the same `location`, and the first `agent` must carry the `tool` in order to pass it.
- `walk`: Walks from a `location` to another. Both locations must be linked to allow the `agent` to walk between them.
- `hang`: Hangs a `picture` on a certain `location` with a `tool`. Both the `agent` and the `picture` must be placed at the same `location`, and the `agent` must carry the `tool` in order to hang the `picture`.

The centralized problems based on this domain are defined by introducing the `worker` type. The objects of this type play the role of the agents in the centralized approach.

5.3.1.3 Port domain

This domain models the management of merchandise in a commercial port. When a ship reaches the port, the ship's crew should unload the containers carried by the ship and load it with new cargo. As for the tiers, they store the containers unloaded from the ships. Both the ships and the tiers have a set of slots, in which they can stack the containers. However, the maximum height of the slot is limited, so the the containers can be stacked up to a certain number. Each tier's containers should be arranged properly, in order to have the top-priority containers on top of the different stacks. The ship and the tier exchange container through a conveyor, which can only transport a container simultaneously.

Therefore, the domain is modeled through a set of tier and ship agents. Ship agents are in charge of loading containers into the vessel and unloading them into the conveyor. To unload a container into a conveyor, it must not have any other container on top of it. Therefore, ship agents have the following capabilities:

- `load`: Picks up a `container` from the conveyor and loads it into one of the ship's slots. If the `slot` has some containers yet, the ship agent stacks the `container` over the rest of containers on the stack. The resulting stack should not exceed the maximum height defined for the `slot`.

- `unload`: Picks up a `container` (which does not have any other `container` on top) from one of the ship's slots and loads it into one of the conveyors. If the `container` is stacked over another `container`, it unstacks it. The `conveyor` should be free in order to perform the operation.

Each tier agent has a hoist on its disposal in order to organize the containers in the conveyors. Their objective is to organize the conveyors in order to place the goal containers (containers which have a high priority) on top of the different stacks. The tier agents can perform the following actions:

- `pickUp`: Obtains a `container` from a `conveyor` and stores it in a `slot`. If the `slot` is not free, the `container` is stacked on top of the stack.

- `takeOut`: Picks up a `container` (or unstacks it, if it is on top of another `container`) and puts it down over a `conveyor`. The `container` should be on top of its `slot` and the `conveyor` should be free in order to perform the operation.
- `stack`: Stacks a `container` over another `container` or puts it down over an empty `slot`. The `hoist` should be carrying the `container` and the resulting stack should not exceed the maximum height of the `slot`.
- `unstack`: Lifts a `container` with the `hoist`. The the lift `lift` should be free and the `container` should be on top of a stack (or be the only `container` on the `slot`).
- `checkGoal1`: Checks if a `goal-container` is on top of its stack. If so, the `goal-container` is marked as ready.
- `checkGoal2`: Checks if a `goal-container` is placed just below of another `goal-container`, which is ready. If so, the `goal-container` situated below is also marked as ready.

The centralized problems based on this domain are defined through the introduction of the `worker` type, which has two subtypes, `tier` and `ship`. The objects of these subtypes will represent the agents in the centralized domain.

5.3.2 Tests and results

The following subsections outline the experimental results obtained. We have carried out three different tests. The first one compares the two state-of-the-art POP heuristics presented in section 4.4.3 through a set of single-agent planning tasks based on the domains presented in the previous sections.

The second set of tests compares the quality of the solution plans obtained through a Single-Agent Planning perspective and through our Multi-Agent Planning (MAP) approach. To do so, we have defined a set of MAP tasks and an equivalent centralized version for each one of them.

Finally, we have measured the robustness and scalability of the MAP system by executing a planning task several times, increasing each time the number of planning agents in the system.

5.3.2.1 POP heuristics

This first test compares the efficiency of the MAX and SUM heuristics introduced in section 4.4.3. The experimental DTG heuristic has been excluded

from the test because of its current sub-par performance. The testbed includes five different planning tasks of increasing difficulty for each of the planning domains presented above.

Problem	MAX heuristic				SUM heuristic			
	IDA*		A*		IDA*		A*	
	#Exp	#A	#Exp	#A	#Exp	#A	#Exp	#A
Transport1	32	8	37	8	61	8	37	8
Transport2	2294	10	1827	10	861	10	519	10
Transport3	1385	11	1388	11	2235	11	1954	11
Transport4	6732	12	9679	12	16655	12	7273	12
Transport5	18177	14	10446	14	4488	14	1511	14
Picture1	301	8	277	8	95	8	84	8
Picture2	673	8	1323	8	1163	8	370	8
Picture3	4448	10	3723	9	328	10	158	10
Picture4	18350	11	18197	11	764	11	795	11
Picture5	50809	11	90205	11	17467	11	9612	12
Port1	1207	6	3116	6	39	8	70	6
Port2	3524	8	14778	8	107	8	73	8
Port3	78619	7	63005	7	40	7	59	7
Port4	†	-	200000 †	-	283	11	221	11
Port5	†	-	200000 †	-	342	11	427	11

Table 5.1: POP heuristics comparison

Table 5.1 presents the results of this test. *#Exp* indicates the number of expanded plans in each test, while *#A* refers to the number of actions of the solution plan found. Each planning problem has been tested twice, using an Iterative Deepening A* and an A* search strategy. A dagger (†) indicates either that the system has run out of memory (in an A* search) or that the search process has expanded more than ten million plans (in an IDA* search).

The results show that the performance of the SUM heuristic is vastly superior in terms of expanded plans, particularly in the case of the *port* domain. The *picture* domain presents also better results in the case of the SUM heuristic. The MAX heuristic only presents competitive results in the *transport* problems, showing even better performance than the SUM heuristic in some cases.

Both heuristics obtain similar results in terms of the plan quality; solution plans obtained with both heuristics have the same number of actions in almost all the cases.

In conclusion, the SUM heuristic obtains similar results than the MAX heuristic and reduces significantly the expansion of plans. Therefore we have adopted the SUM function as the heuristic that will guide both the POP search process and the evaluation of refinements in the following tests.

5.3.2.2 Multi-Agent vs. Single-Agent Planning

The second set of tests compares the quality of the solution plans obtained by our MAP framework with the ones generated by a centralized approach. The testbed includes 5 different planning tasks (five tasks per each of the previously defined planning domains) of increasing difficulty. Each planning task has been defined both in a MAP and a centralized fashion. MAP planning tasks have been tried out with our MAP system using the SUM heuristic and an A* search strategy. As for centralized tasks, they have been solved by our MAP system configured as a single-agent POP, using again the SUM heuristic and an A* search process.

Problem	Multi-Agent Planning			Single-Agent Planning	
	#Ag	#Actions	#Time steps	#Actions	#Time steps
Transport1	2	14	10	15	15
Transport2	3	11	9	16	16
Transport3	3	9	8	8	8
Transport4	4	12	10	13	13
Transport5	4	10	7	11	11
Picture1	2	6	6	6	6
Picture2	3	9	9	9	9
Picture3	3	8	8	8	8
Picture4	4	8	6	7	6
Picture5	4	8	3	8	3
Port1	2	6	5	6	6
Port2	2	4	3	4	4
Port3	2	9	7	9	9
Port4	3	3	3	3	3
Port5	3	4	3	4	4

Table 5.2: Single-Agent vs. Multi-Agent Planning comparison

Table 5.2 shows the obtained results. #Ag indicates the number of agents that undertake the planning task in the MAP tests. #Actions and #Time

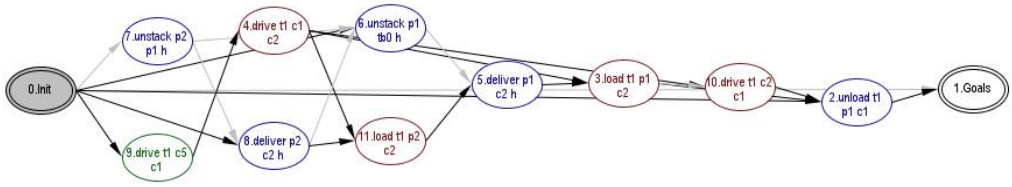

Figure 5.5: Solution plan for the Transport5 MAP task

steps refer to the number of actions and *time steps* of the devised solution plan (notice that we do not take into account the plans' fictitious actions).

As it can be observed, the MAP approach obtains slightly better results in terms of the number of actions of the solution plans. In particular, the *transport* and *port* domains show better results for the MAP approach, while the *picture* domain presents similar results for both planning approaches.

However, the biggest difference comes in terms of the *time steps*. The time steps take into account the actions that can be performed simultaneously in order to measure the time units necessary to execute the plan. For instance, figure 5.5 depicts the solution plan for the Transport5 MAP task. Although the plan is composed by ten planning actions (without taking into account the fictitious ones), it can be executed in only seven time steps, since some of its actions can be executed in parallel.

The MAP approach enforces this parallelism, since different planning entities devise different parts of the plan that can be executed at the same time. The centralized approach is not as effective at introducing parallel actions, as it can be observed in the results of both the *transport* and *port* domains. In these domains, the MAP approach obtains a lower number of time steps in almost all the tests. The *picture* domain, however, is prone to completely linear solution plans, and therefore, the time steps of the solution plans for both approaches coincide in all the tests.

In conclusion, while being a more costly approach (see next section for scalability tests), MAP obtains better solution plans in terms of both actions and time steps.

5.3.2.3 Scalability analysis

This latter test evaluates the scalability of our MAP framework i.e. how the number of agents in the MAP system affects its efficiency. To do so, five different tests have been prepared for the *transport* and the *picture* domains. Each test increases the number of agents by one, keeping the rest of the planning task's parameters unchanged.

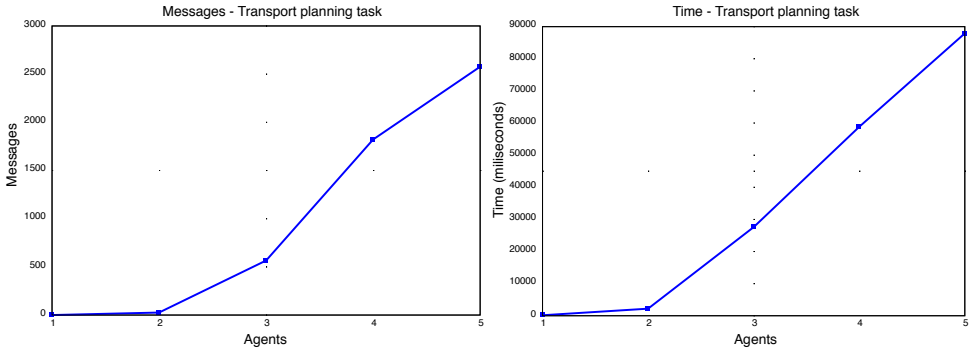

Figure 5.6: Scalability results for the transport domain

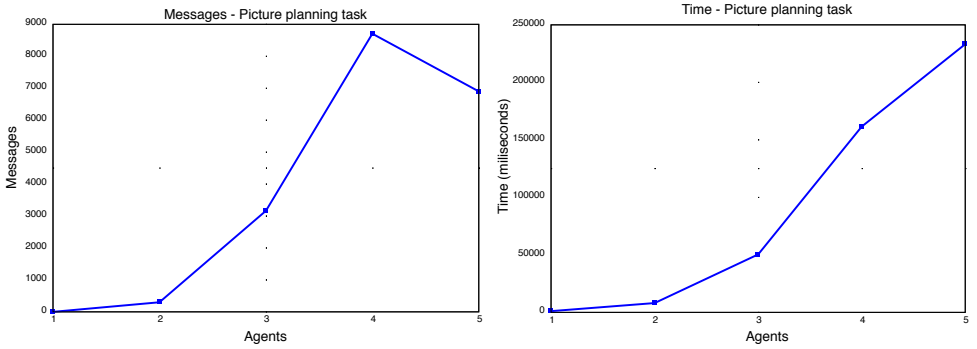

Figure 5.7: Scalability results for the picture domain

Each *transport* domain test includes four cities per area, one truck, one table per depot loaded with three different products, and one package of raw material. The goal defined for all the test problems is to transport two different products.

As for the *picture* domain, all the test problems include three tools and eight different locations. The objective in all the defined problems is to hang three different pictures.

Figures 5.6 and 5.7 depict the results for each domain. As it can be observed, the execution time experiments a notable increase with each new agent included in the MAP process. So does the number of messages exchanged among agents.

These results are caused by the growing number of refinement plans devised by the agents. On the one hand, refinement plans are communicated to all the agents in the MAP system. Therefore, the addition of a planning agent represents an important increase in the number of messages exchanges. On the other hand, each extra planning agent is also able to propose new

refinements, that have to be communicated to the rest of agents. These extra refinements can also be adopted as base plans, increasing this way the complexity of the search tree and the number of messages exchanged in the system.

In conclusion, the number of agents in the MAP system is a parameter that has a notable influence on its efficiency, as the message exchange among agents constitutes one of the bottlenecks of the system. Therefore, one of the challenges we face focuses on tuning up the system performance in order to reduce as much as possible the number of messages exchanged by the agents. This way, we will improve the scalability of the system, giving it the ability to deal with more complex planning tasks.

Chapter 6

Conclusions and future work

This latter chapter summarizes the main contributions presented in this work, outlines the future lines of research and list the related publications produced as a result of this research work.

6.1 Summary of contributions

In the present dissertation, we have introduced a Multi-Agent Planning (MAP) model based on Partial-Order Planning (POP). More precisely, our model follows a refinement planning approach i.e. it is based on the progressive refinement of partial-order plans by the planning agents.

As well as outlining our MAP model, we have detailed the design of a MAP system built upon our refinement planning approach. The system is composed by a set of agents, each of them provided with planning capabilities, that are coordinated through a democratic leadership protocol, in which a different agent takes the leadership role in each iteration of the process.

The problem-solving algorithm is divided in two stages, namely the coordination stage, in which agents exchange proposals and select the next base plan, and the planning stage, in which agents try to refine the current base plan individually, using the partial-order planner they possess.

The POP paradigm has been extended to cope with the new requirements that emerge in the MAP context. We also have designed a customized planning language, that introduces a set of constructs according with these new MAP requirements. We have also performed an early design of a novel POP heuristic, based on the domain transition graph constructs, which creates an estimated scheme of the solution plan, and relies on it in order to guide the search process.

This work also summarizes the details of the development of the MAP

platform. The implementation is fully based on Java language. The development of the multi-agent component of the system relies on the Magentix2 [FAS+10] platform, a FIPA-compliant [ON98] Multi-Agent System. The POP component, integrated in all the planning agents, presents a modular, flexible and extensible implementation that allows the user to integrate new heuristics and search methods. All the components on the system have been built by following the OSGi [All03] standards, which allows to integrate them easily into a service-oriented platform.

Finally, the experimental results show that the new POP heuristic, even in an early stage, appears to be a promising approach, since its performance is comparable to some of the state-of-the-art POP heuristics. We have also carried out some tests to compare the efficiency of the MAP system against centralized versions of the MAP domains. The results reveal that the MAP approach focuses on distributing the workload among the agents, minimizing this way the time steps of the plan through the parallelization of the tasks, rather than the number of actions, as happens in the centralized examples. Hence, the MAP approach presents some advantages over the centralized planning that make its use preferable in many planning applications.

6.2 Future work

The implementation of the MAP system is an important milestone in our work, since this framework will be used as a base to integrate the future developments. Our future lines of research follow two different directions: on the one hand, we will undertake some changes to the new heuristic in order to improve it over the most recent works. On the other hand, we have another research line based on the definition on an argumentation framework for MAP. Our aim is to integrate it into the coordination stage. This way, our MAP system will use a social mechanism by which agents will deliberate to select refinements, as well as having a reliable internal search process to build the individual refinements.

6.2.1 Improvements over the POP heuristic

The heuristic function for Partial-Order Planning (POP) we have designed presents some promising results. Even at an initial stage of its development, its performance is not far from the most recent POP heuristics. The use of the Domain Transition Graphs and the Plan Transition Graph (PTG) allows to obtain valuable information to guide the search process.

However, the heuristic still present some shortcomings. On the one hand,

at this point the heuristic is not able to distinguish between refinements of the same partial plan. For this reason, its performance in depth-first based search algorithms, such as the Iterative Deepening A*, is worse than expected. On the other hand, the heuristic is very dependent on the quality of the Plan Transition Graph. This could lead to poor results in case the PTG is not an accurate estimation of a solution plan.

Hence, the future improvements over the POP heuristic follow two different directions:

- The heuristic calculation algorithm requires further improvements. Currently, it only establishes correspondences between the actions on the partial plan and the PTG. However, it does not value the quality of the actions individually, which would be important to be able to distinguish among refinements of the same partial plan. The causal links and the ordering constraints on the plan and the PTG can also provide valuable contextual information to improve the heuristic. Hence, the whole heuristic will be subject of research over the next months, in order to improve its efficiency.

- The construction of the PTG is also a key aspect of our heuristic. Changes over the construction algorithm alter radically the structure of the PTG. Hence, we will carry out some research on the PTG building process, in order to improve the quality of the PTGs.

6.2.2 Integration of an argumentation framework for coordination

In the context of our Multi-Agent Planning (MAP) system, we have devised an argumentation-based to tackle the multi-agent evaluation and selection of the refinement plans from a social standpoint. Our aim is to enhance the role of argumentation as a means to attain a collective behavior when devising a joint plan. Since agents' decisions are influenced by the other agents' plans, it becomes relevant the use of mechanisms for persuading an agent to adopt a certain course of action, or negotiating on the use of scarce resources. Instead of using a heuristic function to evaluate the different alternatives, agents discuss them through a dialectical process, giving out opinions on the adequacy of these proposals and modifying them to the benefit of the overall process.

Our argumentation framework adapts the instantiation of an argument scheme and the associated critical questions to a MAP context by following the computational representation of practical argumentation presented in

[ABCM06, ABC07]. This approach is particularly suitable for representing multi-agent concurrent plans.

Argumentation will replace the current ad-hoc heuristic function for both validating plan refinements and choosing the most suitable one as the next base plan. Hence, the MAP model will be enhanced by the inclusion of a social approach in which the agents evaluate the plan according to their preferences and interests and effectively discuss on the details of the plan, instead of performing the evaluation through a common heuristic function.

Currently, the argumentation model is at a theoretical stage. One of our research lines for the next months consists on the development and improvement of the argumentation model, in order to integrate it with the MAP system.

The theoretical argumentation model has received generally positive feedback, and has been published in several international conferences. The following section list the research papers written and published in the context of the work presented in this dissertation.

6.3 Related publications

The following research papers are directly related to the present work and have been published during its development:

- O. Sapena, A. Torreño, E. Onaindia. *On the Construction of Joint Plans through Argumentation Schemes.* 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011).

- O. Sapena, E. Onaindia, A. Torreño. *On the use of Argumentation in Multi-Agent Planning.* 19th European Conference on Artificial Intelligence (ECAI 2010).

- E. Onaindia, O. Sapena, A. Torreño. *Argumentation-based Planning in multi-agent systems.* Negotiation and argumentation in Multiagent Systems. Bentham eBooks. In press (2011).

- E. Onaindia, O. Sapena, A. Torreño. *Cooperative Distributed Planning through Argumentation.* International Journal of Artificial Intelligence. ISSN: 0974-0635. In Press (2010).

- A. Torreño, E. Onaindia, O. Sapena. *Reaching a common agreement discourse universe on Multi-Agent Planning.* 5th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2010).

- S. Pajares, E. Onaindia, A. Torreño. *An architecture for Defeasible-Reasoning-based Cooperative Distributed Planning.* 9th International Conference on Cooperative Information Systems (CoopIS 2011).

Bibliography

[ABC07] K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence*, 171:855–874, 2007.

[ABCM06] Katie Atkinson, Trevor J. M. Bench-Capon, and Peter McBurney. Computational representation of practical argument. *Synthese*, 152(2):157–206, 2006.

[All03] O.S.G. Alliance. *OSGi service platform, release 3.* IOS Press, Inc., 2003.

[ASF+95] J. F. Allen, L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light, N. G. Martin, B. W. Miller, M. Poesio, and D. R. Traum. The trains project: a case study in building a conversational planning agent. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:7–48, 1995.

[BB01] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[BCF+95] A. Barrett, D. Christianson, M. Friedman, K. Golden, C. Kwok, J.S. Penberthy, Y. Sun, and D. Weld. UCPOP user's manual (version 4.0). *Technical Report 93-09-06d, University of Washington*, 1995.

[BEG+92] J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. PÃˆrez, S. Reilly, M. Veloso, and X. Wang. Prodigy 4.0: The manual and tutorial. *Technical report CMU-CS-92-150. Carnegie Mellon University*, 1992.

[BF97] A. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BG01] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[BN95] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[BN09] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, 19(3):297–331, 2009.

[BRR09] A. Belesiotis, M. Rovatsos, and I. Rahwan. A generative dialogue system for arguing about plans in situation calculus. In *Argumentation in Multi-Agent Systems, 6th International Workshop, ArgMAS*, pages 23–41, 2009.

[BRR10] A. Belesiotis, M. Rovatsos, and I. Rahwan. Agreeing on plans through iterated disputes. In *AAMAS*, pages 765–772, 2010.

[BW94] A. Barrett and D. S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[CB03] B. J. Clement and A. C. Barrett. Continual coordination through shared activities. In *AAMAS*, pages 57–64, 2003.

[CDB05] J.S. Cox, E.H. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *AAMAS'05*, pages 821–827, 2005.

[Cle05] B. J. Clement. *Proceedings of the Workshop on Multi-agent Planning and Scheduling.* International Conference on Automated Planning and Scheduling ICAPS-05, 2005.

[dDOW99] M.E. desJardins, E.H. Durfee, C.L. Ortiz, and M.J. Wolverton. A survey of research in distributed continual planning. *AI Magazine*, 20(4):13–22, 1999.

[DKS+00] L. Derek, H. Kautz, B. Selman, B. Bonet, H. Geffner, J. Koehler, M. Brenner, F. Hoffmann, J.and Rittinger, C. Anderson, D. Weld, D. Smith, M. Fox, and D. Long. The aips-98 planning competition. *AI Magazine*, 21:13–33, 2000.

[DL91] E. H. Durfee and V. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE*

Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks, 21(5):1167–1183, 1991.

[Dur99] E. H. Durfee. *Distributed problem solving and planning*, volume In Gerhard Weiss editor, pages 118–149. The MIT Press, San Francisco, CA, 1999.

[dWtMW05] M. de Weerdt, A. ter Mors, and C. Witteveen. Multi-agent planning. an introduction to planning and coordination. In *Handouts of the European Agent Systems Summer School (EASSS-05)*, pages 1–32, 2005.

[Ede03] S. Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:195–238, 2003.

[EHN94] K. Erol, J. Hendler, and D. Nau. UCMP: A sound and complete procedure for hierarchical task-network planning. *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 249–254, 1994.

[ER96] E. Ephrati and J. S. Rosenschein. Deriving consensus in multiagent systems. *Artif. Intell.*, 87(1-2):21–74, 1996.

[FAS+10] R.L. Fogués, J.M. Alberola, J.M. Such, A. Espinosa, and A. Garcia-Fornes. Towards dynamic agent interaction support in open multiagent systems. In *Proceedings of the 2010 conference on Artificial Intelligence Research and Development: Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence*, pages 89–98. IOS Press, 2010.

[FL03] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[FN71] R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[Gef00] H. Geffner. Functional strips: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pages 187–209. Kluwer Academic Publishers, 2000.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[GL05] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. *Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 2005.

[GL06] A. Gerevini and D. Long. Preferences and soft constraints in PDDL3. In *Proceedings of ICAPS*, volume 6, pages 46–53. Citeseer, 2006.

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning. Theory and Practice*. Morgan Kaufmann, 2004.

[Gos00] J. Gosling. *The Java language specification*. Prentice Hall, 2000.

[HB06] J. Hoffmann and R. I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[Hel04] M. Helmert. A planning heuristic based on causal graph analysis. *Proceedings of ICAPS*, pages 161–170, 2004.

[HN01] J. Hoffmann and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *JAIR*, 14:253–302, 2001.

[JFL+01] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[Kam97] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.

[KNHD97] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an adl subset. *Proceedings of the Fourth European Conference in Planning*, pages 273–285, 1997.

[Kov11] D. L. Kovacs. Complete BNF description of PDDL3.1. Technical report, 2011.

[KS96] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th National Conference on AI*, pages 1194–1201, 1996.

[LL92] S. Lander and V. Lesser. Customizing distributed search among agents with heterogeneous knowledge. *Proceedings of the First International Conference on Information and Knowledge Management*, pages 335–344, January 1992.

[McD96] D. McDermott. A heuristic estimator for means-ends analysis in planning. *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, 3:142–149, 1996.

[McD00] D. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[MGH+98] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL: The planning domain definition language. 1998.

[MH69] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[MLB92] T. Moehlman, V. Lesser, and B. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 1(2):161–192, January 1992.

[Mye99] K.L. Myers. Cpef: a continuous planning and execution framework. *AI Magazine*, 20(4):63–69, 1999.

[NK01] X.L. Nguyen and S. Kambhampati. Reviving partial order planning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 459–466. Citeseer, 2001.

[ON98] P.D. O'Brien and R.C. Nicol. Fipa - towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[Ped89] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332, 1989.

[PJP97] M.E. Pollack, D. Joslin, and M. Paolucci. Flaw selection strategies for partial-order planning. *Arxiv preprint cs/9706101*, 1997.

[PSJ98] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic Computation*, 8(3):261–292, 1998.

[PW92] J.S. Penberthy and D.S. Weld. UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[Rei87] R. Reiter. On closed world data bases. In *Readings in nonmonotonic reasoning*, pages 300–310. Morgan Kaufmann Publishers Inc., 1987.

[RN03] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[Sap05] O. Sapena. *Planificación Independiente del Dominio en Entornos Dinámicos de Tiempo Restringido*. PhD thesis, Universidad Politécnica de Valencia, 2005.

[ST95] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73:231–252, 1995.

[TNP09] A.Y. Tang, T.J. Norman, and S. Parsons. A model for integrating dialogue and the execution of joint plans. In *8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 883–890, 2009.

[TPW03] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 721–726. Citeseer, 2003.

[Wel94] D.S. Weld. An introduction to least commitment planning. *AI magazine*, 15(4):27, 1994.

[Wel99] D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[Wil88] D.E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.

[YNH92] Q. Yang, D.S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(4):648–676, 1992.

[ZNK07] J.F. Zhang, X.T. Nguyen, and R. Kowalczyk. Graph-based multi-agent replanning algorithm. In *6th Conference on Autonomous Agents and MultiAgent Systems, AAMAS*, 2007.