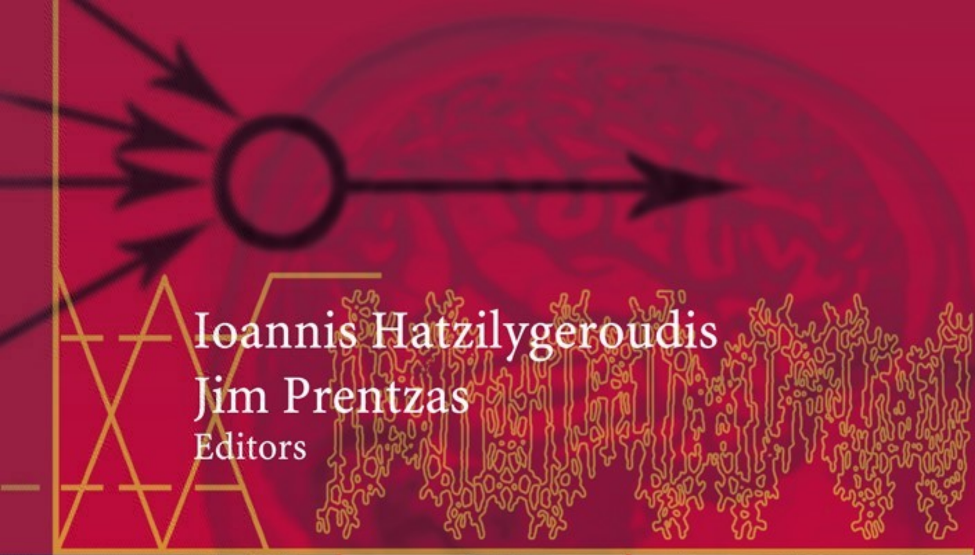

Ioannis Hatzilygeroudis
Jim Prentzas
Editors

SMART INNOVATION, SYSTEMS AND TECHNOLOGIES ■ 8

Combinations of Intelligent Methods and Applications

Proceedings of the 2nd International Workshop, CIMA 2010, France, October 2010

Springer

Ioannis Hatzilygeroudis and Jim Prentzas (Eds.)

Combinations of Intelligent Methods and Applications

Ioannis Hatzilygeroudis and Jim Prentzas (Eds.)

Combinations of Intelligent Methods and Applications

Proceedings of the 2nd International Workshop, CIMA 2010, France, October 2010

Springer

Ioannis Hatzilygeroudis
Graphics, Multimedia & GIS Lab
Department of Computer Engineering &
Informatics
University of Patras
26500 Patras, Hellas, Greece
E-mail: ihatz@ceid.upatras.gr

Jim Prentzas
Democritus University of Thrace
School of Education Sciences
Dept. of Education Sciences in
Pre-School Age, Nea Chili
68100 Alexandroupolis, Greece
E-mail: dprentza@psed.duth.gr

ISBN 978-3-642-19617-1 e-ISBN 978-3-642-19618-8

DOI 10.1007/978-3-642-19618-8

Smart Innovation, Systems and Technologies ISSN 2190-3018

Library of Congress Control Number: 2011924477

Typesetting: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

The combination of different intelligent methods is a very active research area in Artificial Intelligence (AI). The aim is to create integrated or hybrid methods that benefit from each of their components. It is generally believed that complex problems can be easier solved with such integrated or hybrid methods.

Some of the existing efforts combine what are called soft computing methods (fuzzy logic, neural networks and genetic algorithms) either among themselves or with more traditional AI methods such as logic and rules. Another stream of efforts integrates case-based reasoning or machine learning with soft-computing or traditional AI methods. Yet another integrates agent-based approaches with logic and also non-symbolic approaches. Some of the combinations have been quite important and more extensively used, like neuro-symbolic methods, neuro-fuzzy methods and methods combining rule-based and case-based reasoning. However, there are other combinations that are still under investigation, such as those related to the Semantic Web. In some cases, combinations are based on first principles, whereas in other cases they are created in the context of specific applications.

The 2nd Workshop on "Combinations of Intelligent Methods and Applications" (CIMA 2010) was intended to become a forum for exchanging experience and ideas among researchers and practitioners who are dealing with combining intelligent methods either based on first principles or in the context of specific applications.

Important issues of the Workshop were (but not limited to) the following:

- Case-Based Reasoning Integrations
- Genetic Algorithms Integrations
- Combinations for the Semantic Web
- Combinations and Web Intelligence
- Combinations and Web Mining
- Fuzzy-Evolutionary Systems
- Hybrid deterministic and stochastic optimisation methods
- Hybrid Knowledge Representation Approaches/Systems
- Hybrid and Distributed Ontologies
- Information Fusion Techniques for Hybrid Intelligent Systems
- Integrations of Neural Networks
- Intelligent Agents Integrations

- Machine Learning Combinations
- Neuro-Fuzzy Approaches/Systems
- Applications of Combinations of Intelligent Methods to
    - Biology & Bioinformatics
    - Education & Distance Learning
    - Medicine & Health Care

CIMA 2010 was held in conjunction with the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2010). Also, we organized a special track in ICTAI 2010, under the same title.

This volume includes revised versions of the papers presented in CIMA 2010 and one of the short papers presented in the corresponding ICTAI 2010 special track. We have also included a paper of ours as invited paper.

We would like to express our appreciation to all authors of submitted papers as well as to the members of CIMA-10 program committee for their excellent work. We would like also to thank Prof. Eric Gregoire, the ICTAI-10 PC Chair for his help and hospitality.

We hope that these proceedings will be useful to both researchers and developers. Given the success of the first two Workshops on combinations of intelligent methods, we intend to continue our effort in the coming years.

Ioannis Hatzilygeroudis
Jim Prentzas

Workshop Organization

Chairs-Organizers

Ioannis Hatzilygeroudis	University of Patras, Greece
Jim Prentzas	Democritus University of Thrace, Greece

Program Committee

Ajith Abraham	MIR Labs, Europe
Plamen Agelov	Lancaster University, UK
Emilio Corchado	University of Salamanca, Spain
Ronald Denaux	University of Leeds, UK
George Dounias	University of the Aegean, Greece
Artur S. d'Avila Garcez	City University, UK
Elpida Keravnou-Papailiou	University of Cyprus, Cyprus
Constantinos Koutsojannis	University of Patras, Greece
Rudolf Kruse	University of Magdeburg, Germany
George Magoulas	Birkbeck College, Univ. of London, UK
Toni Moreno	University Rovira i Virgili, Spain
Ciprian-Daniel Neagu	University of Bradford, UK
Vasile Palade	Oxford University, UK
David Sanchez	University Rovira i Virgili, Spain
Douglas Vieira	Enacom-Handcrafted Technologies, Brazil

Contents

Defeasible Planning through Multi-agent Argumentation

Sergio Pajares and Eva Onaindia

Abstract. The work reported here introduces DefPlanner, an argumentation-based partial-order planner where different agents that have a partial, and possibly contradictory, knowledge of the world articulate arguments for and against supporting preconditions of the actions to be included in a plan. In this paper, we introduce an extension to multiple agents of the defeasible argumentation formalism that has been proposed to address the task of planning in a single agent environment.

1 Introduction

Planning is the art of building control algorithms that synthesize a course of action to achieve a desired set of goals. The mainstream in planning is that of using heuristic functions to evaluate goals and choices of action or states on the basis of their expected utility to the planning agent [7]. In classical planning, intelligent agents must be able to set goals and achieve them, they have a perfect and complete knowledge of the world, and they assume their view of the world can only be changed through the execution of the planning actions. However, in many real-world applications, agents often have contradictory information about the environment and their deductions are not always certain information, but *plausible*, since the conclusions can be withdrawn when new pieces of knowledge are posted by other agents.

On the other hand, argumentation, which has recently become a very active research field in computer science [2], can be viewed as a powerful tool for reasoning about inconsistent information through a rational interaction of arguments for and against some conclusion. Systems that build on defeasible argumentation apply theoretical reasoning for the generation and evaluation of arguments, and they

Sergio Pajares · Eva Onaindia
Universidad Politécnica de Valencia, Camino de Vera s/n 46022 Valencia, Spain
e-mail: spajares@dsic.upv.es, onaindia@dsic.upv.es

are used to build applications that deal with incomplete and contradictory information in dynamic domains ([11][5][10][12]). Particularly, the application of an argumentation-based formalism to deal with the defeasible nature of reasoning during the construction of a plan has been addressed by Garcia and Simari [13][6].

This paper extends the work of [6] and presents DefPlanner, a defeasible argumentation planner developed for multi-agent environments. We explicitly consider several entities (agents) in the argumentative process for the support of the conditions of a planning action. Some recent works like [16][15] realize argumentation in multi-agent systems using defeasible reasoning but they are not particularly concerned with the task of planning. Specifically, we consider propositional STRIPS planning representation augmented with the incorporation of different sources of defeasible information (agents). Defplanner is a partial-order planner ([1][9]) that invokes an argumentation process where many different agents with different opinions exchange arguments and counterarguments in order to determine whether a given precondition of an action is supported or not, i.e. it can be defeasibly derived or not.

This paper is organized as follows. Next section summarizes the main notions on defeasible logic and partial-order planning. Section 3 elaborates on the use of argumentation during the construction of a partial-order plan. Section 4 presents the defeasible argumentation process in a multi-agent system, and section 5 presents an example of application. Finally, section 6 concludes and presents some future work.

2 Background

2.1 Defeasible Logic

In this section, we summarize the main concepts of the work on Defeasible Logic Programming (DeLP), a formalism that combines Logic Programming and Defeasible Argumentation [5]. The basic elements in DeLP are facts and rules. Let $\mathscr{L}$ denote a set of literals, where a literal h is a fact A or a negated fact $\sim\!A$, and, the symbol $\backsim$ represents the strong negation. The set of rules is divided into *strict rules*, i.e. rules encoding strict consequences, and *defeasible rules*, which derive uncertain or defeasible conclusions. A strict rule is an ordered pair $head \leftarrow body$, and a defeasible rule is an ordered pair $head \prec body$, where $head$ is a literal, and $body$ is a finite non-empty set of literals. For example, the strict rule $animal \leftarrow bird$ is denoting the piece of information "a bird is an animal". However, a defeasible rule is used to describe tentative knowledge that may be used if nothing else can be posed against it, e.g. "birds fly" (fly $\prec$ bird).

Using facts, strict and defeasible rules, an agent is able to satisfy some literal h as in other rule-based systems. Let X be a set of facts in $\mathscr{L}$, STR a set of strict rules, and DEF a set of defeasible rules. A **defeasible derivation** for a literal h from X, denoted as $X \mid\!\sim h$, consists of a finite sequence $h_1,\ldots,h_n = h$ of literals such that h_i is a fact ($h_i \in \mathscr{L}$), or there is a rule in $STR \cup DEF$ with head h_i and body $b_1,\ldots,b_k$, and every literal of the body is an element h_j of the sequence appearing before h_i

($j < i$). A set X is contradictory, denoted $X \mid\!\sim \bot$, if two contradictory literals, eg. h and $\backsim h$, can be derived from X.

In our planning framework, the agent's knowledge base is formed by a consistent set of facts Ψ, and a set of defeasible rules Δ.

Definition 1. *Let h be a literal, and let $\mathscr{K} = (\Psi, \Delta)$ be the knowledge base of an agent. We say that $\langle \mathscr{A}, h \rangle$ is an* ***argument structure*** *for h, or simply* ***argument*** *for h, if $\mathscr{A}$ is a set of defeasible rules of Δ, such that:*

- *there exists a defeasible derivation of h from $\Psi \cup \mathscr{A}$,*
- *the set $\Psi \cup \mathscr{A}$ is non-contradictory, and*
- *$\mathscr{A}$ is minimal, i.e., there is not a $\mathscr{A}' \subset \mathscr{A}$, such that $\mathscr{A}'$ satisfies the above two conditions.*

The literal h is called the conclusion of the argument, and $\mathscr{A}$ the support of the argument.

Definition 2. *Two literals h_1 and h_2* ***disagree*** *iff the set $\Psi \cup \{h_1, h_2\}$ is contradictory. Two complementary literals h and $\sim h$ disagree because for any set Ψ, $\Psi \cup \{h, \sim h\}$ is contradictory. We say that the argument $\langle \mathscr{A}_1, h_1 \rangle$ is in conflict or counter-argues the argument $\langle \mathscr{A}_2, h_2 \rangle$ at the literal h, if and only if there exists a sub-argument $\langle \mathscr{A}, h \rangle$ of $\langle \mathscr{A}_2, h_2 \rangle$, that is $\mathscr{A} \subseteq \mathscr{A}_2$, such that h and h_1 disagree. If $\langle \mathscr{A}_1, h_1 \rangle$ is a counterargument for $\langle \mathscr{A}_2, h_2 \rangle$ at literal h, then h is called a counter-argument point, and the subargument $\langle \mathscr{A}, h \rangle$ is called the disagreement subargument [5].*

In short, two arguments are in conflict if they support contradictory conclusions, or one of the arguments is in conflict with an inner part of the other argument. That is, if the head of a defeasible rule in one of the arguments contradicts the head of a defeasible rule in the other argument.

In order to deal with counterarguments, a central aspect is to establish a formal **comparison criterion** among arguments. A possible preference relation among arguments is the so-called *generalized specificity* [14]. We consider an argument $\mathscr{A}1$ is preferred to an argument $\mathscr{A}2$ if $\mathscr{A}1$ is more precise (it is based on more information), or more concise (it uses fewer rules in the conclusion derivation). In such a case, it is said $\mathscr{A}1$ is more specific than $\mathscr{A}2$. For example, $\langle \{c \prec a, b\}, c \rangle$ is more specific than $\langle \{\sim c \prec \sim a\}, \sim c \rangle$. We use $\langle \mathscr{A}1, h1 \rangle \succ \langle \mathscr{A}2, h2 \rangle$ to denote $\langle \mathscr{A}1, h1 \rangle$ is more specific than $\langle \mathscr{A}2, h2 \rangle$ The preference criterion is needed to decide whether an argument defeats another or not, as disagreement does not imply preference.

Definition 3. *The argument $\langle A_1, h_1 \rangle$ is a* ***defeater*** *for $\langle A_2, h_2 \rangle$ iff there is a subargument $\langle A, h \rangle$ of $\langle A_2, h_2 \rangle$ such that $\langle A_1, h_1 \rangle$ is a counterargument of $\langle A_2, h_2 \rangle$ at literal h, and $\langle A_1, h_1 \rangle \succ \langle A, h \rangle$.*

Definition 4. *An* ***argumentation line*** *for $\langle \mathscr{A}_0, h_0 \rangle$ is a sequence of arguments, denoted $\Lambda = [\langle \mathscr{A}_0, h_0 \rangle, \ldots, \langle \mathscr{A}_m, h_m \rangle]$, where each element of the sequence $\langle \mathscr{A}_i, h_i \rangle$, $i > 0$, is a defeater of its predecessor $\langle \mathscr{A}_{i-1}, h_{i-1} \rangle$. Certain constraints over Λ are considered in [5] in order to avoid several problematic and undesirable situations that may arise in Λ.*

Definition 5. *A* ***dialectical tree*** *for the argument $\langle \mathscr{A}_0, h_0 \rangle$, denoted $\mathscr{T}_{\langle \mathscr{A}_0, h_0 \rangle}$, is defined by the root of the tree, labeled with $\langle \mathscr{A}_0, h_0 \rangle$, and a set of argumentation lines from the root, where every node (except the root) represents a defeater of its parent, and leaves correspond to non-defeated arguments, arguments with no defeaters.*

Some examples of dialectical trees can be found in [5]. In order to decide whether the argument at the root of a given dialectical tree is defeated or not, it is necessary to perform a bottom-up analysis of the tree. Every leaf of the tree is marked *undefeated* and every inner node is marked *defeated*, if it has at least one child node marked undefeated. Otherwise, it is marked undefeated. Let $\mathscr{T}^*_{\langle \mathscr{A}, h \rangle}$ denote a marked dialectical tree of the argument $\langle \mathscr{A}, h \rangle$. A literal h is said to be ***warranted***, if and only if there is an argument $\langle \mathscr{A}, h \rangle$ for h such that the root of the marked dialectical tree $\mathscr{T}^*_{\langle \mathscr{A}, h \rangle}$ is marked undefeated. In such a case, $\langle \mathscr{A}, h \rangle$ is a warrant for h. If a literal h is a fact then h is also warranted as there are no counterarguments for $\langle \emptyset, h \rangle$. Otherwise, if all arguments for h are marked as defeated then the literal h is said to be ***not warranted***.

2.2 Partial-Order Planning

Planning is the art of building control algorithms that synthesize a course of action to achieve a desired set of goals. We consider planning problems encoded in a formal, first-order language such as STRIPS [4], particularly in a propositional version of STRIPS. We will denote the set of all propositions by $\mathscr{P}$ (ground facts or literals). A planning state s is defined as a finite set propositions $s \subseteq \mathscr{P}$. A (grounded) planning task is a triple $\mathscr{T} = \langle \mathscr{O}, \mathrm{i}, \mathscr{G} \rangle$, where $\mathscr{O}$ is the set of deterministic actions of the agent's model that describes the state changes, and $\mathrm{i} \subseteq \mathscr{P}$ (the initial state) and $\mathscr{G} \subseteq \mathscr{P}$ (the goals) are sets of propositions. An action $a \in \mathscr{O}$ is a tuple $a = (pre(a), add(a), del(a))$, where $pre(a) \subseteq \mathscr{P}$ is the set of propositions that represents the action's preconditions, and $add(a) \subseteq \mathscr{P}$ and $del(a) \subseteq \mathscr{P}$ are the sets of propositions that represent the positive and negative effects, respectively. We will represent an action a as follows:

$$\{q_1, \ldots, q_n, \sim r_1, \ldots, \sim r_m\} \xleftarrow{id} \{p_1, \ldots, p_k\} \tag{1}$$

where id is the action name, $\forall_{i=1}^{k} p_i \in pre(a)$, $\forall_{i=1}^{n} q_i \in add(a)$, and $\forall_{i=1}^{m} r_i \in del(a)$. An action a is executable in state s if $pre(a) \subseteq s$. The state resulting from executing a is defined as $s' = (s \setminus del(a)) \cup add(a)$. That is, we delete any proposition in s that belongs to $del(a)$, and add the propositions in $add(a)$. A solution plan (Π) for a planning task $\mathscr{T}$ is a set of actions $\Pi = \{a_1, \ldots, a_n\} \subseteq \mathscr{O}$ such that when applied to i, it leads to a final state in which the goals $\mathscr{G}$ are satisfied. A planning task $\mathscr{T}$ is solvable if there exists at least one plan for it.

In what follows, we provide a brief introduction to the Partial-Order Planning (POP) paradigm ([1][9]). A more detailed tutorial can be found in [17]. In POP, search is done through the space of incomplete partially-ordered plans as opposite to state-based planning. Thus, a key concept in POP is that of *partial-order plan*.

Definition 6. *A* ***partial-order plan*** *is a tuple $\Pi = \langle \mathscr{A}P, \mathscr{O}R, \mathscr{C}L, \mathscr{O}C, \mathscr{U}L \rangle$, where:*

- *$\mathscr{A}P \subseteq \mathscr{O}$ is the set of ground actions*[1] *in Π.*
- *$\mathscr{O}R$ is a set of ordering constraints ($\prec$) over $\mathscr{O}$*
- *$\mathscr{C}L$ is a set of causal links over $\mathscr{O}$. A causal link is of the form (a_i, p, a_j), and denotes that the precondition p of action a_j will be supported by an add effect of action a_i.*
- *$\mathscr{O}C$ is the set of* open conditions *of Π. Let $a_i \in \mathscr{O}$; if $\exists p \in pre(a_i) \wedge \nexists a_j \in \mathscr{O} / (a_j, p, a_i) \subseteq CL$, then p is said to be an open condition.*
- *$\mathscr{U}L$ is the set of* unsafe causal links *of Π, also called the* threats. *Let $(a_i, p, a_j) \subseteq \mathscr{C}L$; (a_i, p, a_j) is unsafe if there exists an action $a_k \in \mathscr{O}$ such that $p \in del(a_k)$ and $\mathscr{O}R \cup \{a_i \prec a_k \prec a_j\}$ is consistent.*

Given a planning task $\mathscr{T} = \langle \mathscr{O}, \mathrm{i}, \mathscr{G} \rangle$, a POP algorithm starts with an empty partial plan and keeps refining it until a solution plan is found. The initial empty plan $\Pi_0 = \langle \mathscr{A}P, \mathscr{O}R, \mathscr{C}L, \mathscr{O}C, \mathscr{U}L \rangle$ contains only two dummy actions $\mathscr{A}P = \{a_0, a_f\}$, the *start* action a_0, and the *finish* action a_f, where $pre(a_f) = \mathscr{G}$, $add(a_0) = \mathrm{i}$, $\{a_0 \prec a_f\} \subseteq \mathscr{O}R$, $\mathscr{C}L = \emptyset$, $\mathscr{O}C = \mathscr{G}$ and $\mathscr{U}L = \emptyset$. The empty plan has no causal links or threats, but, has open condition corresponding to the preconditions of a_f (the top-level goals $\mathscr{G}$). A refinement step in a POP algorithm involves two things; first, selecting a flaw (an open condition or a threat) in a partial plan Π, and then selecting a resolver for the flaw. The different ways of solving a flaw are:

- Supporting an open condition with an *action step*. If p is an open condition, an action a needs to be selected that achieves p. a can be a new action from $\mathscr{O}$, or any action that already exists in $\mathscr{A}P$. Solving an open condition involves adding a causal link to Π to record that p is achieved by the chosen action step.
- Solving a threat with an *ordering constraint*. When the flaw chosen is an unsafe causal link (a_i, p, a_j) that is threatened by an action a_k, it can be repaired either by adding the ordering constraint $a_k \prec a_i$, or the constraint $a_j \prec a_k$, into $\mathscr{O}R$. This solving method involves reordering the action steps in Π.

Definition 7. *A plan $\Pi = \langle \mathscr{A}P, \mathscr{O}R, \mathscr{C}L, \mathscr{O}C, \mathscr{U}L \rangle$ is* ***complete*** *if it has no open conditions ($\mathscr{O}C = \emptyset$).*

Definition 8. *A plan $\Pi = \langle \mathscr{A}P, \mathscr{O}R, \mathscr{C}L, \mathscr{O}C, \mathscr{U}L \rangle$ is* ***conflict-free*** *if it has no unsafe causal links ($\mathscr{U}L = \emptyset$).*

Definition 9. *A plan $\Pi = \langle \mathscr{A}P, \mathscr{O}R, \mathscr{C}L, \mathscr{O}C, \mathscr{U}L \rangle$ is a* ***solution*** *if it is complete and conflict-free.*

[1] Partial-order planners are capable of handling partially instantiated action instances and hence, the definition of a partial order plan typically includes a set of equality constraints on free variables in $\mathscr{O}$ [9]. We will, however, restrict our attention to ground action instances without any loss of generality for our purposes.

3 Argumentation in POP

The task of the agents in classical planning is to be able to set goals and achieve them, i.e. finding a causal chain of actions that, when applied in the initial state, it achieves the desired (sub)goals. In this sense, the set $pre(a)$ of a planning action a is interpreted as a set of *achievable* preconditions. However, actions can also have preconditions whose predicates are not affected by any of the actions available to the planning agent. Instead, the predicate's truth value is the result of a derivation obtained by forward chaining inference rules. More concretely, in our framework, the agent is equipped with a set of planning actions, $\mathscr{O}$, and a knowledge base $\mathscr{K} = (\Psi, \Delta)$ where:

- Ψ is a consistent set of facts. Initially, $\Psi = \mathrm{i}$, and this set will be updated accordingly with the *add* and *del* effects of the applicable actions.
- Δ is a set of *defeasible* rules that will be used to derive plausible information, tentative conclusions that might be withdrawn with new pieces of information.

In conclusion, a planning action a is a tuple $a = (pre(a), add(a), del(a))$, where the set $pre(a)$ is divided into two subsets:

- $pre\_ach(a)$ denotes the set of *achievable* preconditions of the action a. The semantics is the same as in classical planning; an achievable precondition p of an action a is supported if it exists a set of actions from $\mathscr{O}$ that achieves the fact, and p **holds** in the state in which a will be applied, i.e. p is not deleted by any action before it holds in the state.
- $pre\_der(a)$ denotes the set of *derivable* preconditions of the action a, the set of preconditions that can be solved via a defeasible derivation. More particularly, the semantics is that a derivable precondition p of an action a is supported if there exists an argument $\langle A, p\rangle$ such that the root of a the tree $\mathscr{T}^*_{\langle \mathscr{A}, p\rangle}$ is marked undefeated, i.e. p **is warranted** in the state in which a will be applied.

Achievable preconditions are supported in a partial-order plan through action steps (see section 2.2). On the other hand, derivable preconditions are supported through argument steps as proposed in the argumentation-based formalism presented in [6]. Hence, we define a POP paradigm in combination with the argumentation formalism described in section 2.1, and we analyze the interplay of arguments and actions when constructing plans using POP techniques.

Definition 10. *Let $\mathscr{K} = (\Psi, \Delta)$ be the knowledge base of an agent; and let $\langle \mathscr{A}, p\rangle$, $\mathscr{A} \subseteq \Delta$, an argument that supports a derivable literal p. The set $facts(\mathscr{A})$ contains the facts that appear in the bodies of the rules in $\mathscr{A}$.*

In a partial-order plan Π, when an argument $\langle \mathscr{A}, p\rangle$ is used to support a derivable precondition p of an action a_i, Π will contain a new element, a ***support link*** of the form $(\mathscr{A}, p, a_i)$. This refinement step for solving a derivable precondition of an action is called *argument step* [6]. Like causal links, support links are used to support a derivable precondition with the conclusion of an argument. Assuming an argument step $\mathscr{A}1 = \langle \mathscr{A}, p \rangle$, we can interpret that $add(\mathscr{A}1) = \{p\}$, and $pre(\mathscr{A}1) = facts(\mathscr{A}1)$. As can be observed, the introduction of argument steps does not imply any changes in the POP algorithm.

Under this new perspective, we reformulate the definition 6 as follows: A **partial-order plan** is a tuple $\Pi = \langle \mathscr{A}P \cup \mathscr{A}R, \mathscr{O}R, \mathscr{C}L \cup \mathscr{S}L, \mathscr{O}C \cup \mathscr{D}P, \mathscr{U}L \rangle$, where $\mathscr{A}P$, $\mathscr{O}R$, $\mathscr{C}L$, $\mathscr{O}C$ and $\mathscr{U}L$ have the usual meaning, $\mathscr{A}R$ is the set of argument steps included in Π, $\mathscr{S}L$ is the set of support links, and $\mathscr{D}P$ is the set of pending derivable preconditions of the actions in Π. Note that the facts of an argument step are the achievable preconditions of the argument and as such they are included as open conditions in the set $\mathscr{O}C$.

Unlike the approach presented in [6], DefPlanner is a defeasible argumentation-based planner in which many different agents with different opinions argue with each other on the warranty of a given argument. During the plan construction, at the time of solving a derivable precondition p, DefPlanner invokes a procedure and agents initiate a discussion in order to check whether p can be warranted or not. This procedure builds a dialectical tree for each supporting argument of p and finally returns whether p is defeated or undefeated. This *multi-agent discussion* is explained in detail in next section. Hence, in the case of DefPlanner, argument steps are only inserted in a partial-order plan as long as it has been proven the argument is undefeated. This contrasts with other approaches in which each supporting argument gives rise to a different alternative in the POP algorithm, and discussions on the warranty of a given argument take place in case a counter-argument is introduced in the plan. In conclusion, DefPlanner only inserts provably undefeated arguments in a plan, and, consequently, no threats involving two argument steps may appear in our approach. Let $\langle \mathscr{A}1, p \rangle$ be an argument step inserted in a plan Π; if argument $\langle \mathscr{A}2, q \rangle$ is later inserted in Π then DefPlanner guarantees $\mathscr{A}2$ is not a counter-argument of $\mathscr{A}1$ and viceversa.

Additionally, in this first approach of DefPlanner, we assume a piece of information can not be both derived and achieved. That is, a proposition p is either defeasibly derived through a dialectical tree by using the rules in Δ, or achieved through a course of actions in $\mathscr{O}$. Thus, the predicates of defeasible information are never affected by the available planning actions $\mathscr{O}$ and, consequently, no action-argument threats exist. In section 6, we elaborate on this issue for future versions of DefPlanner.

4 Defeasible Argumentation in a Multi-Agent System

DefPlanner implements a Multi-Agent System (MAS) (figure 1) to assist during the construction plan. Agents can adopt one of the four different roles specified in this MAS:

- *Client role*: The user is represented by an agent playing this role, which is in charge of requesting a plan for a given set of goals.
- *POP role*: The agent playing this role, that is, the planner takes as input the set of goals and returns a solution plan that satisfies the client goals. There is only one agent playing the POP role per MAS.
- *Argumentative role*: An agent ag_i which plays this role is associated with a set of defeasible rules representing the tentative information of the agent about the environment (Δ_i). The task of each argumentative agent ag_i is to participate as far as possible in the multi-agent discussions for warranting a given literal. Each agent has an associated utility function[2] that is used to maximize its benefits.
- *Mediator role*: The agent which plays this role (only one per MAS) is in charge of managing the multi-agent argumentation process.

A MAS, as defined in this paper, is formed by a POP agent which reasons about which action step (for solving an open condition), or ordering constraint (for solving a threat) should be chosen in the next iteration of the POP algorithm; a group of non-self-interested argumentative agents, which join together to reason about the argument step that should be chosen to satisfy/warrant a derivable precondition;

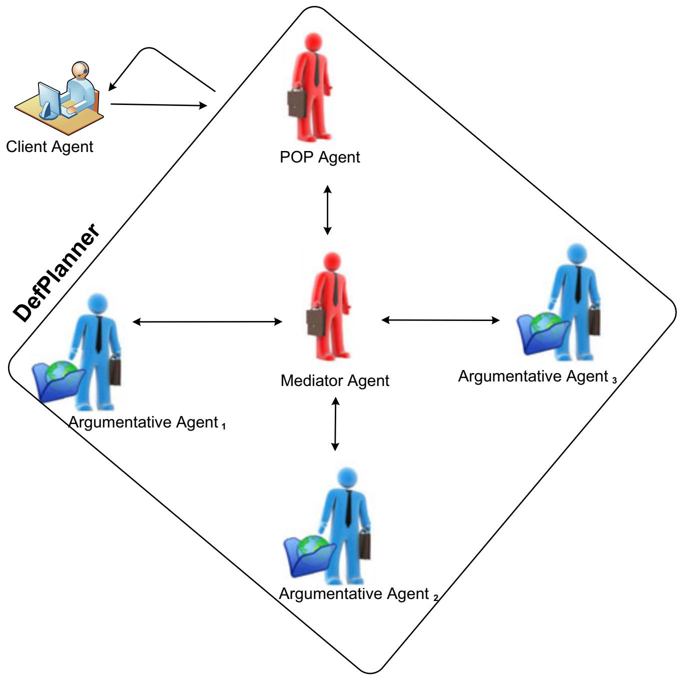

Fig. 1 An overview of DefPlanner.

[2] For instance, in terms of less cost, time, resources or increased safety could be expressed their utility functions.