Journal of Artificial Intelligence Research 51 (2014) 1-70 Submitted 2/14; published 9/14

Cooperative Monitoring to Diagnose Multiagent Plans

Roberto Micalizio MICALIZIO@DI.UNITO.IT
Pietro Torasso TORASSO@DI.UNITO.IT
Dipartimento di Informatica,
Università di Torino
corso Svizzera 185, 10149 - Torino, Italy

Abstract

Diagnosing the execution of a Multiagent Plan (MAP) means identifying and explaining *action failures* (i.e., actions that did not reach their expected effects). Current approaches to MAP diagnosis are substantially centralized, and assume that action failures are independent of each other.

In this paper, the diagnosis of MAPs, executed in a dynamic and partially observable environment, is addressed in a *fully distributed* and *asynchronous* way; in addition, action failures are no longer assumed as independent of each other.

The paper presents a novel methodology, named *Cooperative Weak-Committed Monitoring* (CWCM), enabling agents to cooperate while monitoring their own actions. Cooperation helps the agents to cope with very scarcely observable environments: what an agent cannot observe directly can be acquired from other agents. CWCM exploits nondeterministic action models to carry out two main tasks: detecting action failures and building *trajectory-sets* (i.e., structures representing the knowledge an agent has about the environment in the recent past). Relying on trajectory-sets, each agent is able to explain its own action failures in terms of *exogenous events* that have occurred during the execution of the actions themselves. To cope with dependent failures, CWCM is coupled with a diagnostic engine that distinguishes between *primary* and *secondary action failures*.

An experimental analysis demonstrates that the CWCM methodology, together with the proposed diagnostic inferences, are effective in identifying and explaining action failures even in scenarios where the system observability is significantly reduced.

1. Introduction

Multiagent Plans (MAPs) are adopted in many applicative domains, from Web services to service robots, whenever the interactions among cooperative agents have to be organized in advance (i.e., planned), in order to reach an acceptable efficiency level during the execution; consider for instance, orchestrated Web services (Yan, Dague, Pencolé, & Cordier, 2009), assembling tasks (Heger, Hiatt, Sellner, Simmons, & Singh, 2005; Sellner, Heger, Hiatt, Simmons, & Singh, 2006), or service robotics (Micalizio, Torasso, & Torta, 2006). MAPs are therefore characterized by a cooperative team of agents that perform actions concurrently in order to achieve a common global goal.

The use of MAPs in real-world scenarios, however, has to cope with a critical issue: plan actions can deviate from their expected nominal behaviors due to the occurrence of (unpredictable) *exogenous events*. These deviations are typically considered as *plan failures* since they prevent the agents to reach their goals. Indeed, although MAPs are very versatile

©2014 AI Access Foundation. All rights reserved.

systems, they are also particularly fragile: the failure of an action can easily propagate through the system causing the failures of other actions, even assigned to different agents.

In order to make the execution of a MAP robust (i.e., tolerant to at least some exogenous events), it is therefore important to detect and isolate action failures, and provide a human user (or a plan repair module) with a set of possible explanations for the detected failures. Some recent works (see e.g., Mi & Scacchi, 1993; Gupta, Roos, Witteveen, Price, & de Kleer, 2012; Micalizio, 2013) have argued that a plan repair procedure can be more effective when the causes of the plan failure are known (e.g., identified via diagnosis).

Over the last decade, the problem of diagnosing the execution of a MAP has been faced from different perspectives. Since their seminal work, Kalech and Kaminka (2003) focus on coordination failures and introduce the notion of *social diagnosis*. Social diagnosis relies on an abstract representation of the MAP at hand, given in terms of *behaviors*, and aims at explaining why two or more agents have selected conflicting behaviors. Kalech and Kaminka (2011) and Kalech (2012) present alternative algorithms for inferring a social diagnosis.

Other approaches (see e.g., de Jonge, Roos, & Witteveen, 2009; Roos & Witteveen, 2009; Micalizio & Torasso, 2008, 2009) adopt an explicit representation of a MAP in terms of agents' actions and shared resources. In particular, in their diagnostic framework, Roos and Witteveen (2009) and de Jonge et al. (2009) consider action failures (i.e., actions that do not reach their expected effects), and introduce the notion of *plan diagnosis*. A plan diagnosis is a subset of (already performed) actions that, when assumed abnormal, make the plan execution consistent with the observations received so far. Since the same set of observations can possibly be explained by many plan diagnoses, Roos and Witteveen (2009) present a criterion for identifying preferred diagnoses that is based on the predictive power of these diagnoses.

These proposals, however, rely on some assumptions that might limit their applicability in some real-world scenarios. First of all, they assume some form of synchronization among the agents (e.g., synchronized selection of behaviors, or execution of actions). More importantly, action failures are assumed to be mutually independent. Furthermore, in the particular case of social diagnosis, agents cooperate with each other by exchanging their own belief states, but this might be a critical issue when they have to keep some information private. On the other hand, in the framework proposed by Roos and Witteveen (2009), diagnostic inferences are substantially centralized.

In this paper we aim at relaxing these assumptions by extending the relational-based framework introduced by Micalizio and Torasso (2008, 2009). Similarly to Roos and Witteveen, we adopt an explicit representation of the MAP at hand in term of agents' actions and shared resources. But differently from them, our action models include nominal as well as faulty evolutions. As we will argue in the rest of the paper, this kind of extended action models subsumes the action models proposed by Roos and Witteveen.

In addition, we aim at a fully distributed solution that does not rely on the synchronized execution of the actions (i.e., no global clock is available). A distributed solution to *social diagnosis* was also proposed by Kalech, Kaminka, Meisels, and Elmaliach (2006). In their work, however, a form of synchronization among the agents is required as agents select their next behavior simultaneously. Moreover, the agents cooperate with each other by sharing their own belief states. In our proposal, coordination among the agents is achieved by means of the exchange of direct observations between agents. The idea is that an observation acquired by an agent can be used by other agents in their own reasoning. To understand the difference between exchanging belief states and direct observations, we have to note that a belief state is an interpretation of observations made by a specific agent according to its local knowledge. Since the agent might have a partial knowledge of the environment, its belief states could be ambiguous or even erroneous. Therefore, when agents exchange with each other their own beliefs, they may also propagate their errors. Conversely, when the coordination consists in the exchange of direct observations, agents can infer their own beliefs without the risk of being conditioned by the errors made by others.

We propose a framework that, relying on the notion of Partial-Order, Causal-Link (POCL) Plans (see Cox, Durfee, & Bartold, 2005; Weld, 1994; Boutilier & Brafman, 2001), limits the number of messages exchanged between agents to the number of causal links existing between actions assigned to different agents.

In our proposal, communication plays a central role not only for assuring a consistent execution of the MAP, but also for easing the diagnostic task. We consider that the environment where the agents operate is scarcely observable, so agents can directly acquire information about just a small portion of their surroundings. Dealing with scarce observations can be very challenging for solving the diagnostic task as this situation might prevent the detection of action failures. To cope with this issue we propose in this paper a strategy named *Cooperative Weak-Committed Monitoring* (CWCM), which extends the weak-committed monitoring introduced by Micalizio and Torasso (2009). CWCM allows the agents to cooperate with each other so that an agent can infer the outcome of its own actions (i.e., failure detection) from the pieces of information provided by the other agents.

As soon as failures are detected, these must be diagnosed in order to identify their root causes. In this paper, we propose a local diagnostic process where each agent can diagnose the failure of its own actions without the need of interacting with other agents. In particular, our diagnostic inferences take into account that failures of different actions may be dependent. In other words, an action failure can be an indirect consequence (i.e., a *secondary failure*) of the failure of a preceding action (i.e., *primary failure*). Identifying primary and secondary failures is essential from the point of view of plan repair as primary failures are the root causes for the anomalous observed execution. In principle, a plan repair that recovers from primary failures should also recover from secondary failures. An interesting property of our methodology is that the process of inferring primary and secondary failures is performed autonomously by each agent, just relying on the trajectory-sets built during the cooperative monitoring.

1.1 Contributions

The paper contributes to the diagnosis of MAP execution in many ways. First of all, the paper shows how the *extended action models* proposed for the monitoring purpose can be obtained compositionally from the nominal models of actions, and from the models of the exogenous events that can affect those actions. Thus, a knowledge engineer can take advantage of this by focusing on the models of the elementary components of the systems (e.g., actions and exogenous events), and then creating complex, extended, action models by composing the elementary components.

In addition, the proposed CWCM framework is *fully distributed*: each agent monitors its own actions, and there is no a central agent that traces all actions in progress. Thus, CWCM can be applied in those domains where agents operate in physically distributed environments, and hence a centralized solution could be impractical. Another important feature of CWCM is that it is *asynchronous*: neither assumption on synchronized execution of actions, nor assumptions on how long actions should last are made. In other words, agents do not share a global clock. Of course, some form of synchronization is still necessary when mutual exclusion is required for accessing critical resources. In such cases, however, we will prefer to use the term *coordination*.

CWCM also represents a valid solution whenever a diagnosis of a MAP is performed in environments characterized by scarce observability levels. In fact, a significant contribution of CWCM is its *cooperative monitoring protocol* that enables the agents to acquire information about the system resources from each other. In the paper we argue that the number of messages exchanged via our cooperative protocol is linear in the number of inter-agent causal links (i.e., causal dependencies existing between any pair of agents).

A last important contribution of the paper is the ability of distinguishing between *primary* and *secondary failures*. Previous approaches (see e.g., Micalizio & Torasso, 2008; Roos & Witteveen, 2009), in fact, assume that action failures are independent of each other.

1.2 Outline

The paper is organized as follows. In Section 2 we introduce a basic multiagent planning framework that we use as a starting point of our work. In Section 3 the basic framework is extended by relaxing some important assumptions. Section 4 formally presents the Cooperative Weak-Committed Monitoring (CWCM) strategy, while the local diagnostic inferences are discussed in Section 5. The paper closes with a detailed experimental analysis in Section 6, and a critical discussion of related works in Section 7.

The paper also includes an Appendix where we briefly discuss how CWCM can be implemented by means of Ordered Binary Decision Diagrams (OBDDs) (Bryant, 1992, 1986), and formally analyze the computational complexity of such an implementation.

2. Multiagent Planning: a Basic Framework

This section is organized in three parts. First, we introduce some basic notions on Multiagent Planning and the terminology we use throughout the paper. Then, we discuss how the propositional planning language can be translated into a state-variable representation. Finally, we present a basic strategy for plan execution in multiagent settings in which we highlight the importance of the cooperation among the agents even under the strong assumption that no exogenous event occurs. Such an assumption is relaxed in Section 3.

2.1 Preliminary Concepts on Multiagent Planning

Since we are interested in diagnosing systems that can be modeled as multiagent plans, we begin our discussion by presenting a framework developed within the planning community to represent and synthesize this kind of plans. It is worth noting that the planning problem is typically approached in propositional terms: preconditions and effects of actions are literals that must be true, respectively, before and after the application of the actions themselves. Thus we intuitively introduce in this section some planning notions in propositional terms, however, in Section 2.2, we argue that when addressing the problem of plan execution, it is more convenient to handle a representation of the system in terms of state variables, and hence we translate the propositional framework into a state variables one.

An important assumption holding throughout the paper is that, although the observations gathered by agents at execution time can be partial, they are always correct; we will elaborate more on this point in Section 3.1.

2.1.1 Multiagent Plans

The Multiagent Plan (MAP) systems we take into account in this paper can be modeled as a tuple $\mathcal{S}=\langle \mathcal{T}, RES, P\rangle$, where:

- $\mathcal{T}$ is the team of cooperating agents; agents are denoted by letters i and j;
- RES is the set of shared resources and objects available in the environment; we assume that all the resources are reusable (i.e., they are not consumable), and can only be accessed in mutual exclusion; note that agents can exchange each other pieces of information about these resources, so the space of resource names is a common language through which agents communicate;
- P is a completely instantiated Partial-Order Causal-Link Plan (POCL) (Weld, 1994), resulting from a planning phase as the ones by Boutilier and Brafman (2001) or by Cox et al. (2005). For the sake of simplicity, our MAP P has a simplified structure since it does not involve concurrency and non-concurrency constraints. More precisely, our MAP P is a tuple $\langle I, G, A, R, C\rangle$ where:

  - I is a set of propositional atoms representing the initial state of the system at planning time.
  - G is another set of propositional atoms representing the global goal to be achieved. Note that G is the conjunction of sub-goals G^i each agent $i \in \mathcal{T}$ is in charge of.
  - A is the set of the action instances the agents have to execute; each action a is assigned to a specific agent in $\mathcal{T}$. At planning time, we assume that an action a is modeled just in terms of preconditions $pre(a)$ and effects $eff(a)$, where both are conjunctions of grounded atoms (see PDDL level 1, Fox & Long, 2003). In the rest of the paper, we denote as a^i_l the l-th action performed by agent i.
  - R is a set of precedence links between action instances; the precedence link $\langle a, a'\rangle$ in R means that action a' can only start after the completion of action a.
  - C is a set of causal links of the form $lk : \langle a, q, a'\rangle$; the link lk states that action a provides action a' with service q, where q is an atom occurring both in the effects of a and in the preconditions of a'.

We assume that MAP P:

- is *flaw-free*: a nominal execution of P from I achieves G;

- is *safe* with respect to the use of the resources. Intuitively, we say that a resource $res \in RES$ is used *safely* iff at each execution step, res is either not assigned, or assigned to exactly one agent. This is similar to the *concurrency requirement* (Roos & Witteveen, 2009): two actions are not executed concurrently if they both require the same subset of resources;
- has *no redundant actions*: even though P is not necessarily optimal, it only contains actions that directly or indirectly provide services to help the achievement of the goal. This means that there always exists a chain of causal links between any action in the plan and at least one atom in the goal G.

To guarantee the resource safeness, we introduce the notion of *working sessions* associated with resources and agents:

Definition 1 *Let res be a resource in RES, and i be an agent in $\mathcal{T}$, a working session $ws_{\langle res,i \rangle}$ for i using res is a pair $\langle a^i_o, a^i_c \rangle$ of actions in A^i such that:*

- *a^i_o precedes a^i_c (i.e., $a^i_o \prec a^i_c$, where $\prec$ is the transitive closure of the precedence relations in R).*
- *a^i_o is an action in A^i through which agent i acquires res, this is modeled by specifying the atom $available(res)$ in the preconditions of a^i_o. Moreover, there exists in C an incoming causal link of the form $\langle a^j_k, available(res), a^i_o \rangle$, where a^j_k is an action assigned to agent j (possibly a^j_k can be a_0 i.e., the pseudo action whose effects determine the initial state of the system). Action a^i_o "opens" the working session. No other action $a^i_h \in A^i$ between a^i_o and a^i_c (i.e., $a^i_o \prec a^i_h \prec a^i_c$), has an incoming causal link labeled with service $available(res)$ coming from an action of another agent $j \neq i$.*
- *a^i_c is an action in A^i through which agent i relinquishes resource res in favor of another agent $j \neq i$. This is modeled by means of a causal link $\langle a^i_c, available(res), a^j_k \rangle$ in C, meaning that action a^i_c releases res as one of its effects, and that $available(res)$ is one of the preconditions of a^j_k. Action a^i_c "closes" the working session. Of course, agent i can release resource res to at most one agent; i.e., the outgoing link $\langle a^i_c, available(res), a^j_k \rangle$ is unique. In addition, no action $a^i_h \in A^i$ between a^i_o and a^i_c, has an outgoing link labeled with service $available(res)$ directed towards an action of another agent j.*
- *any action a^i_h between a^i_o and a^i_c can use res; i.e., res can be mentioned in the preconditions and effects of a^i_h. More precisely, a causal link mentioning res between two actions a^i_h and a^i_k in A^i is allowed only if both a^i_h and a^i_k belong to the same working session, namely, $a^i_o \prec a^i_h \prec a^i_k \prec a^i_c$.*

Given a working session $ws_{\langle res,i \rangle}$, we denote its opening and closing actions as $opening(ws_{\langle res,i \rangle})$ and $closing(ws_{\langle res,i \rangle})$, respectively. Two working sessions $ws_{\langle res,i \rangle}$ and $ws'_{\langle res,j \rangle}$ are *consecutive*, $ws_{\langle res,i \rangle} \lhd ws'_{\langle res,j \rangle}$, if $closing(ws_{\langle res,i \rangle})$ provides $opening(ws'_{\langle res,j \rangle})$ with service $available(res)$.

Proposition 1 *A MAP P satisfies the resource safeness requirement if for each resource $res \in RES$, all the working sessions over res can be totally ordered in a sequence $ws^1_{\langle res,i_1 \rangle} \lhd ws^2_{\langle res,i_2 \rangle} \ldots \lhd ws^n_{\langle res,i_n \rangle}$, for any agent $i_j \in \mathcal{T}$ (with $j : 1..n$).*

This means that, independently of the agents accessing the resource, two working sessions over the same resource res never overlap each other. Possibly, an agent i can have more than one session in the sequence, meaning that the agent acquires and relinquishes resource res many times along the plan.

The *resource safeness* requirement is an extension of the concurrency requirement introduced by Roos and Witteveen (2009). In fact, while the concurrency requirement implicitly imposes an ordering between two actions that cannot be performed simultaneously, the resource safeness imposes an ordering between blocks of actions identified by a working session. This is necessary in order to model situations where an agent uses a set of resources for a number of consecutive actions that cannot be interleaved with the actions of other agents. A working session is a sort of critical section that cannot be interrupted. It is worth noting that, since a working session is associated with a single resource, and since there is no constraint between the working sessions of two resources, it is possible that an action $a^i_c \in A^i$ closes two different working sessions. For example, let $ws_{\langle res,i\rangle}$ and $ws'_{\langle res',i\rangle}$ be two working sessions of agent i using resources res and res', respectively; then it is possible that an action a^i_c is both $closing(ws_{\langle res,i\rangle})$ and $closing(ws'_{\langle res',i\rangle})$. In addition, the resource res could be released in favor of agent j, while resource res' could be released in favor of another agent k. This is modeled through the two causal links $\langle a^i_c, available(res), a^j_x\rangle$ and $\langle a^i_c, available(res'), a^k_y\rangle$.

2.1.2 Local Plans

Since we are interested in a fully distributed framework for both plan execution and plan diagnosis, we impose that every agent just knows the portion of P it has to perform; we thus introduce the notion of *local plan*. Intuitively, a local plan P^i is the projection of P over the actions assigned to agent i; P is therefore partitioned into as many local plans as there are agents in $\mathcal{T}$. More formally, given an agent $i \in \mathcal{T}$, the local plan P^i assigned to i is a tuple $P^i = \langle I^i, G^i, A^i, R^i, C^i_{local}, C^i_{in}, C^i_{out}\rangle$, where I^i represents the portion of the initial state known by agent i when the plan execution starts; G^i is the sub-goal assigned to agent i; A^i, R^i and C^i_{local} have the same meaning as A, R and C, respectively, restricted to the actions assigned to agent i. The remaining sets, C^i_{out} and C^i_{in}, highlight the causal dependencies existing between the actions of agent i and actions assigned to other agents in $\mathcal{T}$: C^i_{out} maintains all the outgoing causal links modeling services that agent i provides other agents with; whereas, C^i_{in} maintains all the incoming causal links modeling services that agent i receives from other agents. To simplify the plan structure, precedence links between actions of different agents are not allowed in R. This, however, is not a real limitation as precedence links between actions of different agents could be expressed as causal links in which the exchanged service refers to a "dummy" resource.

In the rest of the paper we will consider a local plan P^i as a partially ordered set of actions. However, we assume that each agent can perform just one action per time, so in the rest of the paper we index the actions according to their execution step. In other words, P^i will be executed by i as a sequence $\langle a^i_0, a^i_1, \ldots, a^i_n, a^i_\infty\rangle$, where a^i_0 and a^i_∞ are two pseudo-actions similar to the ones introduced by Weld (1994). Action a^i_0 has no preconditions and its effects coincide to the initial state I^i known by agent i; in particular, this pseudo-action is also used to determine the initial set of resources assigned to agent i: any link leaving from a_0^i and labeled with service $available(res_k)$ denotes that res_k is assigned to agent i. Action a_∞^i, on the other hand, has no effects and its preconditions correspond to sub-goal G^i assigned to i.

2.2 Translating the Propositional Framework into a State-Variable Representation

Although most of the planning approaches in literature relies on the propositional language to represent planning problems, we adopt in this paper a representation based on multi-valued state variables which is similar to the SAS$^+$ approach introduced by Jonsson and Bäckström (1998). The reason for this choice stems from the fact that a multi-valued variable implicitly encode mutual exclusion constraints among the values of its domain: the variable can just assume one value at a given time. Thus, it is easier to represent the evolution of the system state over the time as a sequence of assignments of values to the state variables. This solution has also been effectively adopted for the diagnosis of plans (Roos & Witteveen, 2009), and as Helmert (2009) has proven, it is not restrictive since it is always possible to translate a propositional representation into a set of multi-valued state variables. In the rest of the section we briefly describe how the propositional planning language (see e.g., Nebel, 2000) can be mapped to the state variables representation. Three main aspects are addressed: (1) how to represent agents' states in terms of state variables rather than sets of propositional fluents; (2) how to represent the exchange of services among agents; and (3) how to model actions in terms of state variables.

2.2.1 Mapping Atoms to Variables

From our point of view, both action models and system states can be represented in terms of a finite set of multi-valued state variables, each of which has a finite and discrete domain. Given the MAP system $\mathcal{S}=\langle \mathcal{T}, RES, P\rangle$, we associate all agents in $\mathcal{T}$ and resources in RES with a set of state variables; each of these variables maps a subset of atoms of the corresponding propositional representation.

It follows that the current system state is given by the values currently assigned to the state variables of all agents and resources. Such a global view, however, is inadequate to achieve a fully distributed framework. We thus introduce the notion of *agent state*, which just captures the portion of the system state relevant for a specific agent i in the team. Each agent $i \in \mathcal{T}$ is associated with a set VAR^i of variables. Each variable $v \in VAR^i$ has a finite and discrete domain $dom(v)$. A state S_l^i for agent i at a given execution step l is therefore an assignment of values to the variables in VAR^i. More precisely, $S_l^i(v) \in dom(v)$ is the value assumed by variable $v \in VAR^i$ in the agent state S_l^i. A partial agent state σ_l^i is an assignment of values to just a subset of variables in VAR^i.

Variables in VAR^i are partitioned into two subsets: END^i and ENV^i. Set END^i includes endogenous state variables (e.g., the agent position); whereas ENV^i includes variables about the shared resources. Because of the partitioning of the global system state into agent states, the state variables of the resources are duplicated into as many copies as there are agents in $\mathcal{T}$. Therefore, for each resource $res_k \in RES$, there exists a private copy res_k^i belonging to ENV^i. To maintain the consistency among these private copies, we rely on two assumptions: (1) MAP P is flaw-free, and (2) P satisfies the resource safeness requirement. These two assumptions induce on variables in ENV^i an implicit constraint: at each execution step only one agent i knows the actual state of a given resource res_k. As a consequence, only the private copy res^i_k keeps the correct value. For all the other agents, resource res_k is "out of sight"; this is modeled by assigning the value *unknown* to their local variables about res_k; namely, for each $j \neq i$, res^j_k =*unknown*. Thus the consistency among the different private copies is implicitly maintained.

Of course, when agent j acquires resource res_k from agent i, it also comes to know the actual state of that resource. The values of variables in ENV^i can in fact be exchanged among the agents. In Section 2.3 we present a basic cooperative protocol that enables the agents to share their knowledge about the resources while preserving the resource safeness requirement; such a basic protocol will be later extended in Section 4.5. Variables in END^i, on the other hand, refer to agent i, and in our framework cannot be shared with other agents.

2.2.2 Mapping Services to Variables Assignments

Since we adopt a representation based on state variables, also a service exchanged between two agents is conveniently modeled as a value assignment to a resource variable. For instance, the causal link $lk : \langle a^i, available(res_k), a^j\rangle$ is used in the propositional representation to model that action $a^i \in A^i$ provides action $a^j \in A^j$ with resource res_k. The same link can be rewritten in the state variables representation as $\langle a^i, res_k = available, a^j\rangle$, where res_k is the name with which agents i and j refer to a specific resource in RES. In other words, res_k is a sort of meta-variable that is known both by agent i and agent j. Of course, each agent maps the meta-variable res_k into its own private copy. More precisely, by means of the link $\langle a^i, res_k = available, a^j\rangle$, agent i is able to communicate j a change in the state of resource res_k: agent j knows that, after the execution of a^i, its private variable res^j_k has the value *available*, meaning that j can now use res_k. It is worth noting that *available* is just a special value, used by agents for exchanging resources. In general, agents communicate each other domain-dependent values about the state of a resource (e.g., the position of a block in the blocks world domain).

Relying on the state variables representation, we can identify the set of resources available to a given agent i at the l-th plan execution step as $availRes^i(l) = \{res_k \in RES | S^i_l(res^i_k) = available\}$. In the next subsection we focus on a coordination protocol that allows agents i and j to exchange information about the shared resources.

2.2.3 Mapping Propositional Action Models to Function-Like Models

Let Q be a subset of the state variables VAR^i, in the rest of the paper we denote with $\Sigma(Q)$ the space of possible assignments to the variables in Q, and with $\mathbf{Q}$ we denote a specific assignment in $\Sigma(Q)$; that is, a specific partial state for agent i. In the rest of the paper we use *premises*(a^i_l) and *effects*(a^i_l) to denote the subset of status variables in VAR^i over which preconditions and effects, respectively, of action a^i_l are defined. Thus **premises**(a^i_l) represents a specific assignment of values in the space $\Sigma(premises(a^i_l))$. Note that the set of premises includes also those services that must be provided by other agents, and which therefore correspond to incoming causal links for action a^i_l. Similarly, the set of effects includes also those services that agent i provides other agents with, encoded as causal links outgoing from a^i_l.

Given an action instance $a^i_l \in A^i$, the deterministic (nominal) model of a^i_l is a mapping:

$$f^{nom}_{a^i_l} : \mathbf{premises}(a^i_l) \rightarrow \mathbf{effects}(a^i_l)$$

where $\mathbf{premises}(a^i_l)$ is an assignment of variables in $premises(a^i_l) \subseteq VAR^i$ representing the preconditions of a^i_l, and $\mathbf{effects}(a^i_l)$ is an assignment of variables in $effects(a^i_l) \subseteq VAR^i$ modeling the effects of the action. We also assume that $\mathbf{effects}(a^i_l) \subseteq \mathbf{premises}(a^i_l)$; this is not a substantial limitation, however. Any variable v, that in principle would only be mentioned in the effects of an action model, can also be mentioned in the premises of the action by allowing v to assume any possible value of its domain. The reason for this assumption will be clear in Section 3.2 where we formalize the notion of exogenous events.

Since we are interested in the execution of plans, we reformulate the applicability of an action to a state (see Nebel, 2000) in terms of executability. Given an agent state S^i_l, an action instance $a^i_l \in A^i$ is *executable* in $S^i_l \in \Sigma(VAR^i)$ iff $S^i_l \models \mathbf{premises}(a^i_l)$; indeed, this strong condition will be relaxed in the next section. Using the same terminology by Roos and Witteveen (2009), we can say that action a^i_l is executable in S^i_l only if a^i_l is *fully enabled* in S^i_l. The result of the execution of a^i_l enabled in S^i_l is the new agent state S^i_{l+1} in $\Sigma(VAR^i)$, called *successor state*:

$$S^i_{l+1} = \begin{cases} (S^i_l \setminus effects(a^i_l)) \cup \mathbf{effects}(a^i_l) & \text{if } S^i_l \models \mathbf{premises}(a^i_l) \text{ and} \\ & S^i_l \not\models \bot \text{ and } \mathbf{effects}(a^i_l) \not\models \bot, \\ \emptyset, & \text{otherwise.} \end{cases}$$

More precisely, $S^i_l \setminus effects(a^i_l)$ is a partial agent state obtained by removing from S^i_l all variables mentioned in $effects(a^i_l)$. Such a partial state is subsequently completed with the new assignments in $\mathbf{effects}(a^i_l)$, yielding a new (complete) agent state S^i_{l+1}. This state transformation, however, is legal only when: (1) action a^i_l is fully enabled in S^i_l (i.e., $S^i_l \not\models \bot$ and $S^i_l \models \mathbf{premises}(a^i_l)$), and (2) the action effects are consistent (i.e., $\mathbf{effects}(a^i_l) \not\models \bot$). Otherwise, the new agent state is undefined.

2.3 Plan Execution Under Nominal Conditions

The actual execution of a MAP requires some form of coordination among the agents. The decomposition of the global plan into local plans, in fact, allows the agents to execute their actions in a fully distributed way without the intervention of a global manager: agents execute actions *concurrently* and *asynchronously* (i.e., no global clock is required). In addition, differently from previous approaches (see e.g., de Jonge et al., 2009; Micalizio & Torasso, 2008), where actions just take a single time slot to be completed and their execution is globally synchronized, we do not make any assumption on the duration of each action. Agent coordination is therefore essential in order not to violate the precedence and causal constraints introduced during the planning phase. For the sake of clarity, we first present a basic coordination strategy by assuming that:

BaDE(P^i=$\langle I^i, G^i, A^i, E^i, C^i_{local}, C^i_{in}, C^i_{out}\rangle$)
$l \leftarrow 1$
$S^i_l \leftarrow I^i$
while $a^i_l \neq a^i_\infty$ **do**
  $S^i_l \leftarrow$ *consume-inter-agent-messages(inbox)*
  if a^i_l is *executable* in S^i_l **then**
    execute a^i_l
    $S^i_{l+1} \leftarrow f^{nom}_{a^i_l}(S^i_l)$
    $obs^i_{l+1} \leftarrow$ gather observations
    if $S^i_{l+1} \cup obs^i_{l+1} \models \perp$ **then**
      stop execution and propagate failure
    end if
    for each causal link $lk \in C^i_{out}$ such that $lk : \langle a^i_l, v = d, a^j_m\rangle$ **do**
      notify agent j about the achievement of service $v = d$
      if $v \in RES$ and $d = available$ **then**
        $S^i_{l+1}(v) \leftarrow unknown$
      end if
    end for
    $l \leftarrow l + 1$
  end if
end while

Figure 1: Basic Distributed Plan Execution (BaDE) strategy.

- *action-based observability*: even though the agents do not have a complete view of the environment, each agent is always able to observe the preconditions and the effects of the actions it performs. In addition, observations are assumed reliable and correct. We denote as obs^i_{l+1} the set of observations received by agent i at the l-th plus 1 execution step after the execution of the l-th action. The observations obs^i_{l+1} can be thought of as sets of value assignments to the variables in $\mathit{effects}(a^i_l)$. That is, for each variable $v \in \mathit{effects}(a^i_l)$, an assignment $v = d$ belongs to obs^i_{l+1}; where $d \in dom(v)$;
- *deterministic actions*: actions can never fail and their models precisely define how the agent state changes;
- *static environment*: the environment can change only as a consequence of the execution of the agents actions.

These three strong assumptions will be relaxed in Section 3, where we will present a more complex coordination strategy that guarantees a consistent access to the resources even when actions can fail.

2.3.1 Basic Coordination Protocol

The coordination protocol we adopt is very simple but effective, and exploits the causal links in P^i. Each outgoing link in C^i_{out} models, in fact, the exchange of a service/resource between agent i - the *service provider* - and another agent j - the *service client*. In order to support the communication among the agents, we assume that each agent has an *inbox*, i.e., a folder where messages coming from the other agents are stored. Whenever i executes an action a^i_l having outgoing links in C^i_{out}, i sends a message for each outgoing link of a^i_l notifying the receiver that the needed service/resource is now available. Likewise, each link in C^i_{in}

models the exchange of a service/resource in which i is the receiver and another agent j is the provider. Whenever i has to execute an action having incoming links in C^i_{in}, it waits for a message for each of these links since the action becomes executable only when all the required services/resources are provided. Under the assumptions made about the MAP P, this protocol guarantees that resources are always accessed consistently. In fact, by assumption P satisfies the resource safeness requirement, which assures that working sessions over the same resource are totally serialized; in particular, two working sessions $ws_{\langle res,i\rangle} \lhd ws'_{\langle res,j\rangle}$ are serialized by means of a causal link $\langle closing(ws_{\langle res,i\rangle}), res = available, opening(ws'_{\langle res,j\rangle})\rangle$. Therefore, when $ws_{\langle res,i\rangle}$ closes, agent i notifies agent j that resource res is now available.

2.4 Basic Distributed Plan Execution (BaDE) Strategy

The high-level plan execution strategy performed by each agent i in the team is outlined in Figure 1. The strategy consists of a loop that iterates as long as there are actions in P^i to be executed. The first step in the loop is to acquire new information from other agents about the shared resources. To accomplish this task, agent i plays the role of *client* of our coordination protocol and gathers all the notification messages, if any, sent by other agents; these notification messages (i.e., the value assignments they convey) are asserted within the current agent state S^i_l so that i updates its local view about the system status and acquires (if available) new resources. After this step, the next action a^i_l is selected: if a^i_l is not executable yet, agent i keeps waiting for further notification messages (i.e., some services/resources are still missing). Otherwise, the agent executes a^i_l in the real world, and then estimates the successor state S^i_{l+1} by exploiting the nominal model $f^{nom}_{a^i_l}$. (Note that the actual execution of a^i_l in the real world may take some time which is not necessarily known in advance.)

Once the action a^i_l has been completed, the agent gathers observations about the effects of the action and matches these observations with the estimated successor state S^i_{l+1}. Since we are assuming that actions cannot fail, no discrepancy between observations and estimations can be discovered. We include the control in line 9 for compatibility with the extension described in sections 3 and 4, where actions may fail.

After the execution of a^i_l, agent i plays the role of *provider* in our coordination protocol and propagates the (positive) effects of action a^i_l towards other agents by sending a notification message for each outgoing link of a^i_l in C^i_{out} (see lines 13 through 16 of the algorithm in Figure 1). In particular, in case agent i has just released a resource ($v \in RES$), the agent sets its private copy about that resource to *unknown*, in this way the resource becomes unreachable to i and the mutual exclusive access is guaranteed.

The last remark about the basic strategy regards the increment of counter l. Note that l is only incremented when an action is executed; thus l does not correspond to a metric time, but it is a pointer to the next action to be performed. Adhering to the BaDE strategy, an agent can conclude its own local plan within a finite amount of time.

Proposition 2 *Let $\mathcal{S} = \langle \mathcal{T}, RES, P\rangle$ be a MAP system such that P is* flaw-free *and satisfies the* resource safeness requirement. *If all agents in $\mathcal{T}$ follow the BaDE strategy in Figure 1, then P is successfully completed in a finite amount of time.*

Proof: It is sufficient to show that at least one action of P is always in execution, at each step, until the achievement of the goal. By contradiction, let us assume that the goal has not been reached, and no action is in progress. Since we assume here that no failure can occur, this situation can happen when agents are in deadlock, waiting for services no one will provide them with. A deadlock may arise either because of an erroneous planning phase, but then P is not flaw-free as initially assumed, or because the BaDE coordination strategy is erroneous. Let us show that when all agents adopt the BaDE strategy, and no failure occurs, the agents never wait indefinitely for accessing a resource. Since P satisfies by assumption the resource safeness requirement, it must hold that $ws_{\langle res,j \rangle} \lhd ws_{\langle res,i \rangle}$; i.e., $closing(ws_{\langle res,j \rangle})$ provides $opening(ws_{\langle res,i \rangle})$ (i.e., a^i_l) with service $res = available$. Agent i waits for service $res = available$ only in two situations: (1) agent j has not performed action $closing(ws_{\langle res,j \rangle})$ yet (correct behavior), or (2) agent j has already performed action $closing(ws_{\langle res,j \rangle})$, but has not sent the appropriate message to i. This second case contradicts the hypotheses that the agents adopt the BaDE coordination protocol. In fact, as required by the protocol, whenever a working session $ws_{\langle res,j \rangle}$ closes, agent j has to send a message to the next agent accessing that resource.

Therefore, if agents are never in deadlock, at least one action is always in execution, and since P has a finite number of actions, the goal G must be achieved within a finite amount of time. $\square$

Example 1. We conclude the section by briefly exemplifying the concepts introduced so far. In particular, we present here the office-like domain we used as a test scenario for our experiments (Section 6), this domain is similar to the ones adopted by Micalizio (2013) and Steinbauer and Wotawa (2008). In this domain, robotic agents deliver parcels to the desks of a number of clerks. A robot can carry one or two parcels depending on whether they are heavy or light, respectively. Figure 2 shows the office-like environment we used for our tests; it includes: 9 desks, distributed over 5 office-rooms, which are connected to each other by means of 8 doors. Moreover, two repositories contain 12 parcels to be delivered (8 light and 4 heavy). Parcels, repositories, doors, and desks are all critical resources as they can be used and accessed by only one agent per time. The domain also includes three parking areas, these are locations where more agents can be positioned simultaneously as they are not critical resources. (The term location is used to identify either parking areas or resources where agent can be physically positioned; e.g., parcels are not locations.)

Agents can perform the following actions: *move* from a location to another, *load*/*unload* parcels within resources which are locations (i.e., not in parking areas), in addition, we impose that no more than one parcel can be positioned on a desk, while repositories have unlimited capacity. Finally, agents can *carry* one heavy parcel or two light parcels from a location to another.

Figure 3 shows a simple example of MAP in our office domain. The team involves three agents `A1`, `A2`, and `A3`, whose plans are in given in three columns, respectively. At the bottom of the picture the effects of pseudoaction a_0 represent the initial states of the three agents. At the top of the picture, the premises of pseudoaction a_∞ represent the desired final state. The objective of the MAP in the figure is to deliver *parcel1* to *desk3* (i.e., an agent has to unload *parcel1* when positioned in *desk3*), and then to bring the parcel back to its initial position in repository *repos1*. Similarly, *parcel2* has first to be delivered to *desk6*,

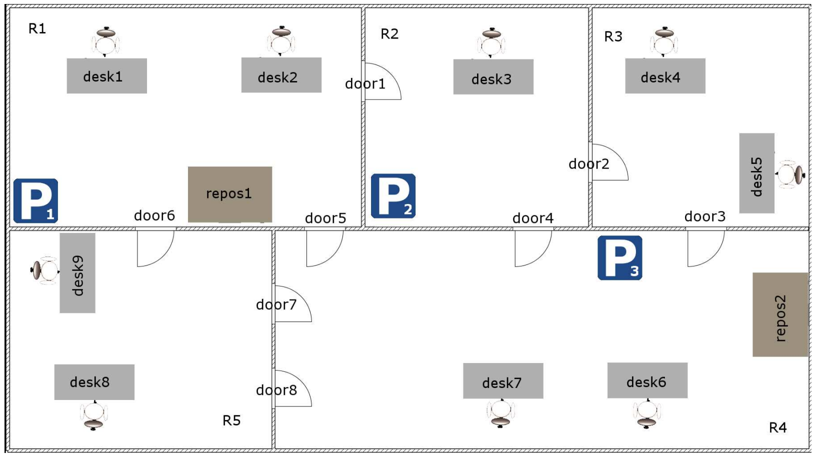

Figure 2: The office-like environment used for the experiments: five rooms R1-R5, two repositories: repos1 and repos2, eight doors, nine desks, and three parking areas.

and brought back to repository *repos2*; while *parcel3*, that has already been delivered to *desk3*, has to be delivered to *desk4*. To ease the readability of the picture, we only show the inter-agent causal links. We use two different graphical notations to distinguish between causal links giving access to resources (diamond-headed), and the causal links that model other kinds of services (black-circle-headed). For instance, the link between actions a_3^1 and a_4^2 is diamond-headed, this means that action a_3^1 provides a_4^2 with service $desk3 = available$ (i.e., after a_3^1, agent `A1` has no longer access to $desk3$). The three dashed rectangles in the picture represent the working sessions associated with resource $desk3$, which is used by the three agents at different execution steps. (The working sessions for the other resources have not been highlighted to avoid the picture becoming too confused.)

Black-circle-headed links are used to represent all the other services. For instance, the link between actions a_2^1 and a_5^2 (labeled as ①) encodes the service *desk3.content=empty*, required by action a_5^2 since at most one parcel can be located on a desktop. The link labeled as ② (from a_5^2 to a_7^3) encodes two services: *desk3.content=parcel1* and *parcel1.pos=desk3*. Similar services are encoded by link ③, but they refer to *desk6* and *parcel2*. ■

3. Extending the Framework

In the previous section we have described a simple coordination strategy that guarantees the consistent execution of a MAP P when three strong assumptions hold: (1) each agent has an *action-based observability*: it can precisely observe the preconditions and the effects of the actions it performs; (2) the environment is *static* (no exogenous events are permitted); and (3) the actions are *deterministic* (no deviation from the nominal behavior is possible). Henceforth we extend the basic framework by relaxing these three assumptions and, as a consequence, by increasing the complexity of the strategy for controlling the distributed plan execution.

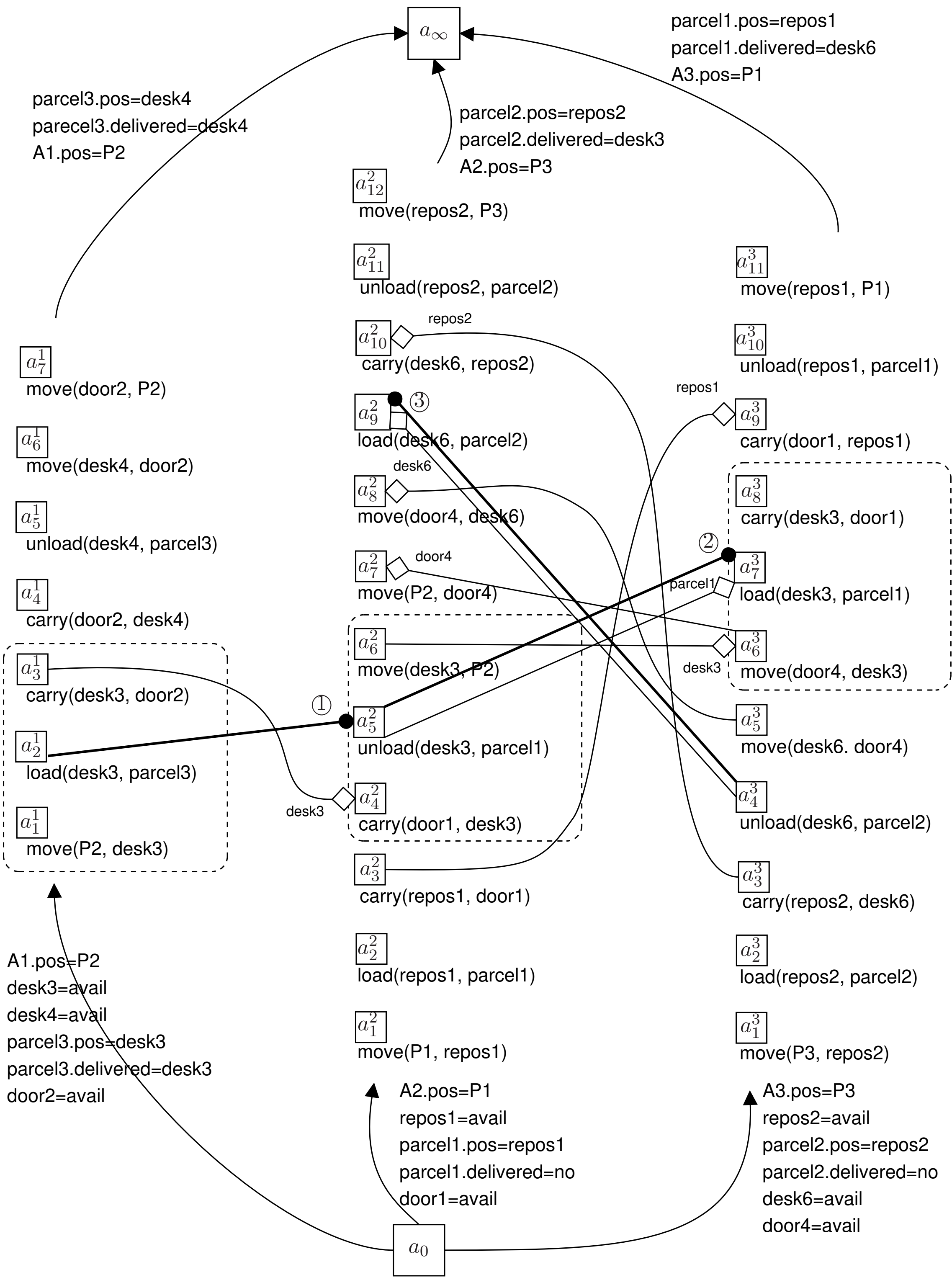

Figure 3: A simple example of MAP in the office-like domain used for testing.

3.1 Partial Observability

The first assumption we relax is the *action-based observability*. In the basic framework, the observations obs^i_{l+1} agent i receives at the $(l+1)$-th step of execution cover all the variables in $effects(a^i_l)$. In the extended framework, obs^i_{l+1} becomes just *partial* since only a subset of variables in $effects(a^i_l)$ is covered in general, and possibly obs^i_l can even be empty. Also in this case we assume that the observations are correct, meaning that the actual state of an agent cannot be inconsistent with the observations received by the agent itself. However, observations can be ambiguous in the sense that for a given variable, an agent just receives a disjunction of possible values. In addition, to guarantee the termination of the plan execution phase, we assume that each agent observes at least the achievement of the atoms in its local goal $G^i \subset G$.

3.2 Plan Threats

The second extension is about the dynamics of the system: we move from a *static* system to a *dynamic* one. This means that, besides the changes due to the agents' actions, the system can also change as a consequence of exogenous, unpredictable events, which typically represent *plan threats* (Birnbaum, Collins, Freed, & Krulwich, 1990) for the nominal execution of the plan. Intuitively, a plan threat can be seen as an abrupt change happened in the environment (i.e., resources), or in the state of an agent.

In this paper we associate the occurrence of an exogenous event to the execution of an action. In other words, an exogenous event can only occur during the execution of an action and can only affect the active variables of that action; namely, the variables mentioned within the premises and the effects of the action. Thus, an exogenous event cannot affect simultaneously two or more actions, but it can have indirect effects on many actions, even of different agents, by means of the shared resources.

In principle, given an exogenous event ξ, one could define a model to predict how ξ will affect the execution of an action. In real-world domains, however, it is not always possible to precisely know in advance the actual impact of an exogenous event: on the one hand, ξ may have non-deterministic effects; on the other hand, not all the effects of ξ may be known. To take into account the possibly non-deterministic effects of exogenous events, we model an exogenous event ξ as a relation $happens_\xi$ defined as follows:

$$happens_\xi \subseteq \Sigma(affectedby_\xi) \times \{\xi\} \times \Sigma(affectedby_\xi) \tag{1}$$

where $affectedby_\xi \subseteq VAR^i$, and $\Sigma(affectedby_\xi)$ is the space of partial agent states defined over $affectedby_\xi$. It is worth noting that $\Sigma(affectedby_\xi)$ can only be empty when $affectedby_\xi$ is empty, too. This enables us to state that, when an exogenous event occurs, then a variable in $affectedby_\xi$ must necessarily evolve unexpectedly. Thus, each tuple in the relation $happens_\xi$ represents a non-deterministic effect of ξ; namely, each tuple represents a possible abrupt change in some agent's status variables.

To deal with not known effects of an exogenous event ξ, we extend the domain $dom(v)$ of each variable $v \in VAR^i$ with the special value *unknown* which leaves open any possible evolution for v.

We denote as $\mathcal{X}$ the set of all exogenous events which might occur during the plan execution. Note that $\mathcal{X}$ also includes a pseudo event ϵ modeling the absence of abrupt changes. Only for this special event it must hold that $\mathit{affectedby}_\epsilon = \emptyset$. Since an exogenous event ξ is defined as a state transition over agents' state variables, ξ can only affect an action a_l^i iff

$$\mathit{affectedby}_\xi \subseteq \mathit{effects}(a_l^i). \tag{2}$$

Namely, ξ affects a subset of the variables over which action a_l^i is defined.

Given an action a_l^i, $\mathcal{X}(a_l^i)$ denotes the subset of exogenous events in $\mathcal{X}$ which satisfy relation (2). Note that, the pseudo event ϵ is always included in $\mathcal{X}(a_l^i)$, for any action $a_l^i \in A^i$, since $\mathit{affectedby}_\epsilon$ (i.e., the empty set) trivially satisfies relation (2).

3.3 Extended Action Models

The last extension we propose is about the action models. Since plan threats occur during the execution of actions, their effects combine with the actions' effects. To estimate how the system evolves over time, it is essential to extend the nominal action model in order to encode, in a single piece of knowledge, the nominal as well as the anomalous evolutions of an action. Intuitively, such an extended model should describe how an agent state S_l^i evolves into a new agent state S_{l+1}^i when agent i has carried out an action instance a_l^i, and when, during the execution of the action, an exogenous event $\xi \in \mathcal{X}$ has occurred, possibly just ϵ. Moreover, in the basic framework we give for granted that an action is performed when it is fully enabled. In our extended framework this condition is not necessarily satisfied. Due to the partial observability, in fact, agent i may be unable to precisely determine whether its next action is fully enabled or not. To cope with this situation, we introduce in Section 3.4 the concept of *possibly enabled* action. For the time being, we just anticipate that an agent may decide to perform an action even when that action is not fully enabled, and hence the extended action model must be so rich to estimate how the state of agent evolves even in such a situation.

The extended model $\mathcal{M}(a_l^i)$ of action a_l^i is derived from the nominal model $f_{a_l^i}^{nom}$, given in terms of premises and effects, and from the set of exogenous events $\mathcal{X}(a_l^i)$; it is formally defined as:

$$\mathcal{M}(a_l^i) : \langle f_{a_l^i}^{nom}, \mathcal{X}(a_l^i), \Delta(a_l^i), \Gamma(a_l^i) \rangle,$$

where $f_{a_l^i}^{nom}$ and $\mathcal{X}(a_l^i)$ have already been introduced; whereas $\Delta(a_l^i)$ and $\Gamma(a_l^i)$ are two transition relations between partial states in $\Sigma(\mathit{premises}(a))$ and in $\Sigma(\mathit{effects}(a))$, through which it is possible to predict how the execution of action a_l^i changes the state of the environment (i.e., the resources held by i) and of agent i itself.

Relation $\Delta(a_l^i)$ estimates the next agent's states when action a_l^i is *fully enabled* in a state S_l^i. This relation results from the combination of the nominal action model with the models of the exogenous events in $\mathcal{X}(a_l^i)$:

$$\Delta(a_l^i) = \bigcup_{\xi \in \mathcal{X}(a_l^i)} \{ f_{a_l^i}^{nom} \odot \mathit{happens}_\xi \} \tag{3}$$

Intuitively, $f_{a_l^i}^{nom} \odot \mathit{happens}_\xi$ is a set of tuples of the form $\langle \mathit{pre}, \xi, \mathit{eff} \rangle$, where pre equals $\mathbf{premises}(a_l^i)$, and $\mathit{eff} \in \Sigma(\mathit{effects}(a_l^i))$ models the abrupt changes caused by event ξ to the nominal effects of action a^i_l. Formally, for each happening $\langle \sigma, \xi, \sigma' \rangle$ in $happens_\xi$ it holds:

$$f^{nom}_{a^i_l} \odot happens_\xi = \bigcup_{\langle \sigma,\xi,\sigma' \rangle \in happens_\xi} \begin{cases} \mathbf{premises}(a^i_l) \times \{\xi\} \times \{(\mathbf{effects}(a^i_l) \setminus \mathit{affectedby}_\xi) \cup \sigma'\} \\ \quad \text{if } S^i_l \models \mathbf{premises}(a^i_l) \text{ and } \mathbf{premises}(a^i_l) \models \sigma; \\ \\ \emptyset \quad \text{otherwise.} \end{cases} \tag{4}$$

It is important to note that, since ϵ is always part of $\mathcal{X}(a^i_l)$, the nominal model $\langle \mathbf{premises}(a^i_l)$, ϵ, $\mathbf{effects}(a^i_l) \rangle$ is always included in $\Delta(a^i_l)$. In particular, in such a state transition, no variable can assume value *unknown*. This follows directly by: (1) the nominal model $f^{nom}_{a^i_l}$ cannot mention the *unknown* value by definition, and (2) the exogenous event ϵ cannot affect any variable since $\mathit{affectedby}_\epsilon$ is empty. Thus, the operation $f^{nom}_{a^i_l} \odot happens_\epsilon$ just reproduces the nominal behavior.

In addition, note that $\mathcal{X}(a^i_l)$ can also include a special exogenous event $\xi_\star$. This symbol denotes an indefinite exogenous event for which no model is given, and hence all variables in *effects*(a^i_l) are mapped to *unknown*: after the occurrence of $\xi_\star$ no prediction is possible.

Relation $\Gamma(a^i_l)$ has the same structure as $\Delta(a^i_l)$ in terms of preconditions, effects and exogenous events, but represents a dual version of $\Delta(a^i_l)$ since it is defined when a^i_l *is not executable* in S^i_l. In fact, $\Gamma(a^i_l)$ is defined in all those states where action a^i_l is not enabled. Let $\Sigma(premises(a^i_l))$ be the space of assignments of values to the variables in $premises(a^i_l)$, $\Gamma(a^i_l)$ is defined over the space of states $\tilde{\Sigma}(premises(a^i_l))$=$\Sigma(premises(a^i_l)) \setminus \mathbf{premises}(a^i_l)$ as:

$$\Gamma(a^i_l) \subseteq \tilde{\Sigma}(premises(a^i_l)) \times \{\xi_\star\} \times \langle unknown, \ldots, unknown \rangle, \tag{5}$$

where $\xi_\star$ denotes an indefinite exogenous event as in Δ. Note that $\Gamma(a^i_l)$ is a weaker model than $\Delta(a^i_l)$ since it invariably assigns the *unknown* value to each variable in *effects*(a^i_l). That is to say, whenever an action is performed in a wrong configuration, its impact on the *effects*(a^i_l) variables becomes unpredictable. Although we use the same symbol $\xi_\star$ to denote indefinite events occurring in $\Delta(a^i_l)$ and in Γ, they have slightly different meanings from a diagnostic point of view that will be discussed in detail in Section 5.

Remark 1. The relational action models we propose are sufficiently flexible to deal with incomplete or imprecise knowledge. In many cases, in fact, it may be too costly (or even impossible) to determine how exogenous events impact the variables in *effects*(a^i_l). The extended framework copes with this problem by allowing three forms of incompleteness:

- The *unknown* value included in the domain of each variable allows to represent that, as an effect of an exogenous event, the value of a variable becomes no more predictable. In the extreme case, all the variables in the effects of an action are set to *unknown* (see the weak model for the exogenous event $\xi_\star$).

- Non-deterministic action evolutions can be defined: an exogenous event may have non-deterministic effects on the states of the agents.

- The weak relation Γ allows us to model the status of an agent after the execution of an action under wrong conditions.

Remark 2. Since actions can be performed even though they are not fully enabled, how can we guarantee that the execution of the plan does not violate the resource safeness requirement? The answer to this question is in the coordination protocol which is part of the Cooperative Weak-Committed Monitoring (CWCM) strategy discussed in Section 4. It is useful to anticipate, however, that the coordination protocol guarantees that an agent uses a resource only when its actions do not violate the resource safeness requirement.

Example 2. Let us consider a simple example from the office domain, and assume that agent `A1` is in charge of performing action `carry(A1, Parc2, desk1, desk2)`; such an action requires `A1` to move from its current position `desk1` to position `desk2` while it is loaded with parcel `Parc2`. The nominal model for such an action can be expressed as the state transition:

$$\langle \texttt{pos = desk1}, \texttt{cObj=Parc2}, \texttt{Parc2pos}_1\texttt{=A1}, \epsilon, \texttt{pos = desk2}, \texttt{cObj=Parc2}, \texttt{Parc2pos}_1\texttt{=A1}\rangle;$$

where `pos` and `cObj` are two endogenous variables for `A1` representing the current position of the agent and the carried object, respectively. The state of shared resource `Parc2` is encoded by variable $\texttt{Parc2pos}_1$, which is the private variable agent `A1` keeps to maintain the position of parcel `Parc2`. For all the other agents, the local copy of variable `Parc2pos` is *unknown*.

The actual execution of the carry action can be affected by a number of exogenous events; for instance, $\xi_{wheels-blocked}$ prevents the agent from moving at all, while $\xi_{wrong-step}$ allows the agent to move, but in a wrong direction. Another event that can affect the carry action is $\xi_{lost-parcel}$: while the agent is moving, the carried object(s) is lost; finally, $\xi_\star$ denotes an unpredictable event occurring when the carry action is attempted in a state in which its preconditions are not satisfied. All these alternative situations are summarized within the extended model showed in Table 1. The first entry of the table is the nominal state transition, the only one labeled with ϵ. Entries from 2 to 5 describe how the action behaves when some known exogenous event occurs. Note that, although the exogenous event is one of the foreseen possibilities, not all its effects may be precisely known; for instance, as an effect of $\xi_{wrong-step}$ and $\xi_{lost-parcel}$ some of the variables assume the value *unknown*. The first five entries of the table represent the Δ relation of the extended model. The last entry of the table, instead, is the Γ relation which allows us to make just weak predictions. The tuple ⟨`pos`=*, `cObj`=*, `Parc2`-*place*=*⟩ is just a shortcut to represent any possible assignment in which the preconditions are not satisfied. Note that, from a practical point of view, it is not necessary to compute this (potentially huge) set explicitly, as we discuss in Appendix A about the implementation. ■

3.4 Extending Some Basic Concepts

Since we have relaxed the three assumptions of the basic framework, we have to review three important concepts: the state of an agent, the executability of an action, and the outcome of an action.

	END		*ENV*		*END*		*ENV*
	`pos`	`cObj`	`Parc2pos`$_1$	event	`pos`	`cObj`	`Parc2pos`$_1$
t_1	`desk1`	`Parc2`	`A1`	ϵ	`desk2`	`Parc2`	`A1`
t_2	`desk1`	`Parc2`	`A1`	$\xi_{wheel-blocked}$	`desk1`	`Parc2`	`A1`
t_3	`desk1`	`Parc2`	`A1`	$\xi_{wrong-step}$	*unknown*	`Parc2`	`A1`
t_4	`desk1`	`Parc2`	`A1`	$\xi_{lost-parcel}$	`desk2`	*empty*	`desk1`
t_5	`desk1`	`Parc2`	`A1`	$\xi_{lost-parcel}$	`desk2`	*empty*	*unknown*
t_6	*	*	*	$\xi_\star$	*unknown*	*unknown*	*unknown*

Table 1: The extended model for the action instance `carry(A1, B2, desk1, desk2)` from the office domain.

3.4.1 Agent's Belief States

First of all, each agent in the team must be able to deal with some form of uncertainty. Since actions may evolve non-deterministically and since the agent cannot observe all the effects of its actions, an agent must be able to deal with *belief states* rather than with agent states. Like an agent state S^i_l, an agent belief state $\mathcal{B}^i_l$ encodes the knowledge agent i has about itself at the l-th execution step. While S^i_l is the precise state assumed by i at step l, $\mathcal{B}^i_l$ is a *set* of possible agent states consistent with the observations received by i. In the rest of the paper we use lowercase s to indicate an agent state among others within a given belief state, while we use uppercase S to indicate the actual agent state at a given execution step. It is important to note that, exactly as an agent state S^i_l, a belief state $\mathcal{B}^i_l$ is defined over all the state variables of agent i; but two states s_1 and s_2 in $\mathcal{B}^i_l$ differ for at least one variable. In other words, there must exists at least one variable $v \in VAR^i_l$ such that $s_1(v)$, the value assumed by v in s_1, is different from $s_2(v)$. Of course, this ambiguity represents an issue in understanding whether the next action a^i_l is executable.

3.4.2 Possibly Enabled Actions

Since we have an agent belief state $\mathcal{B}^i_l$, also the notion of action executability needs to be revised. A very conservative policy would require action a^i_l to be fully enabled in every state s in $\mathcal{B}^i_l$; but due to the partial observability this condition might not be satisfied, so the execution of a MAP could be stopped because no action is enabled even though no plan threat has occurred.

To avoid this situation, we propose a more optimistic policy and introduce the notion of *possibly enabled action*.

Definition 2 Optimistic Policy *Action a^i_l is* possibly enabled *in $\mathcal{B}^i_l$ iff $\exists s \in \mathcal{B}^i_l$ such that a^i_l is fully enabled in s; namely, $s \models$ **premises**(a^i_l).*

It is worth noting that the value *unknown* cannot be used to qualify an action as fully enabled. Such a value, in fact, is used explicitly to state that the agent does not know the value of a variable. Therefore, if variable v has value *unknown* in a state s, and v is also mentioned in **premises**(a^i_l), then a^i_l is not fully enabled in s.

A possibly enabled action is therefore a sort of conjecture: since the action premises are satisfied in at least one state of the belief state, the action is assumed executable. Of course, it may be the case that s, although possible, is not the real state of the agent, and hence the action is performed when its preconditions are not satisfied in the real world.

3.4.3 Action Outcome

In the basic framework we have given for granted that the outcome of an action is always nominal. In the extended framework, however, actions can fail. We individuate three possible action outcomes: the nominal *ok*, the anomalous *failed*, and *pending* for the intermediate situations.

Definition 3 *Action* a^i_l *has outcome* ok *iff* $\forall s \in \mathcal{B}^i_{l+1}$, $s \models \textbf{effects}(a^i_l)$.

That is, the action's effects hold in every state in $\mathcal{B}^i_{l+1}$ (estimated after the execution of a^i_l). Definition 3 does not hold when there exists at least one state $s \in \mathcal{B}^i_{l+1}$ where the nominal effects are not satisfied. In some previous approaches (Micalizio & Torasso, 2008, 2007a), we have introduced and adopted the *strong committed* policy: when the effects of action a^i_l are not satisfied in each state of the belief state $\mathcal{B}^i_{l+1}$, the action has outcome *failed* (see Definition 3). The *strong committed* policy is based on the assumption that, whenever action a^i_l has been successfully completed, agent i receives an amount of observations sufficient to detect the success. Thereby, when the success cannot be detected, a failure must have occurred.

This policy, however, may be unacceptable in some real-world domains where there are no guarantees about the system observability. As a consequence, agent i could infer a failure even when action a^i_l has been completed with success, but the observations are not sufficient to confirm it.

In this paper we define the failure of an action as the dual case of the success:

Definition 4 *Action* a^i_l *has outcome* failed *iff* $\forall s \in \mathcal{B}^i_{l+1}$, $s \not\models \textbf{effects}(a^i_l)$.

Namely, it is not possible to find in $\mathcal{B}^i_{l+1}$ a state s in which all the expected effects of a^i_l have been achieved.

In all those situations where neither the success (Definition 3) nor the failure (Definition 4) can be inferred, the action outcome is *pending*.

Definition 5 *Action* a^i_l *has outcome* pending *iff* $\exists s \in \mathcal{B}^i_{l+1}$, $s \models \textbf{effects}(a^i_l)$ *and* $\exists s' \in \mathcal{B}^i_{l+1}$, $s' \not\models \textbf{effects}(a^i_l)$.

In other words, whenever agent i is unable to determine the success or the failure of action a^i_l, it postpones the evaluation of the action outcome to some step in the future; the action is enqueued into a list $pActs^i$ of pending actions maintained by agent i. We refer to this policy as *weak committed* since the agent does not take decisions whenever there are insufficient observations sufficient to support them. In the next section we discuss the impact of the weak committed policy on the monitoring task.

4. Cooperative Weak-Committed Monitoring

In this section we discuss a fully distributed approach to the problem of monitoring the execution of a MAP. We consider the extended framework previously discussed which introduces two sources of ambiguity: the agent belief states, and the ambiguous action outcomes.

To cope with these forms of uncertainty, we propose a monitoring methodology called *Cooperative Weak-Committed Monitoring* (CWCM), which relies on the weak-committed policy. The CWCM approach allows an agent to detect the outcome of an action some time after its execution. The idea is that the possibly uncertain knowledge an agent has about the environment and itself can be refined over time by exploiting observations that the agent will receive in the future. To get this result, CWCM allows the team members to cooperate with each other during their monitoring tasks.

The rest of this section is organized as follows. We first formalize the notion of *trajectory-set* maintained by each agent, and explain how the extended action models can be used to extend the trajectory-set one step further. Then we discuss how the trajectory-set is refined through the observations and how this helps in determining the outcomes of pending actions (if any). Finally, we redefine the cooperative protocol sketched in the basic framework to obtain a cooperative monitoring protocol. CWCM is entirely formalized in terms of Relational Algebra operators. (For a short introduction to the used operators, see Micalizio, 2013.)

4.1 Trajectory-Set

The weak-committed approach requires that an agent be able to reason about its past. This means that the agent cannot maintain just the last belief state, but it has to keep a *trajectory-set*; i.e., a sequence of belief states that traces the recent history of the agent's state.

We define a trajectory-set as a generalization of an agent trajectory. An *agent trajectory* for agent i, denoted as $tr^i(1,l)$, is defined over a segment $[a^i_1, \ldots, a^i_{l-1}]$ of the local plan P^i, and consists of an ordered sequence of agent states interleaved with exogenous events in $\mathcal{X}$ (including ϵ). An agent trajectory represents a possible evolution of the status of agent i, consistent with the observations the agent has received so far; more formally:

Definition 6 *The* agent trajectory $tr^i(1,l)$ *over the plan segment* $[a^i_1, \ldots, a^i_{l-1}]$ *is*

$$tr^i(1,l) = \langle s_1,\, e_1,\, s_2, \ldots,\, e_{l-1}, s_l \rangle$$

where:

- s_k *($k : 1..l$) is the state of agent i at the k-th step such that* $obs^i_k \cup s_k \not\models \bot$.
- e_h *($h : 1..l-1$) is an event in* $\mathcal{X}(a_h)$ *labeling the state transition from* s_h *to* s_{h+1}.

An agent trajectory is therefore a sequence of agent states, interleaved by events, that traces the agent behavior along a given plan segment. For the sake of discussion, we consider the plan segment as starting from the first performed action a^i_1; in practice, however, the plan segment under consideration can be an intermediate portion of an agent's local plan. We will return on this point in Section 4.6.

Since each state s_k (k in $[1..l]$) is a complete assignment of values to the agent state variables in VAR^i, these variables are duplicated as many times as there are actions in the plan segment under consideration; in the following, we will denote as VAR^i_k the copies of the state variables referring to the k-th execution step.

As noticed above, however, the partial system observability is in general not sufficient for the estimation of a unique trajectory; for this reason agent i keeps a *trajectory-set* $Tr^i[1..l]$, which contains all possible agent trajectories $tr^i(1, l)$ consistent with the observations received during the execution of the plan segment $[a_1^i, \ldots, a_{l-1}^i]$.

Note that, given a trajectory-set $Tr^i[1, l]$, the agent belief state at any execution step k in $[1..l]$ can easily be inferred by projecting $Tr^i[1..l]$ over the state variables VAR_k^i:

$$\mathcal{B}_k^i = \text{PROJECT}_{VAR_k^i}(Tr^i[1..l]) \tag{6}$$

Thus definitions 2 (possibly enabled actions), 3 (successfully completed actions), 4 (failed actions), and 5 (pending action), which are all based on belief states, are still meaningful and do not require to be redefined.

In the rest of the paper, the term *trajectory frontier* (or simply frontier) refers to the last belief state maintained within a trajectory-set. For instance, the frontier of $Tr^i[1, l]$ is the belief state $\mathcal{B}_l^i$. As a general rule, we use l to denote the index of the last execution step (and hence of the frontier); while k is used to refer to a generic execution step in $[1, l]$.

4.2 Extending the Trajectory-Set

The extension of a trajectory-set corresponds to the predictive step of the basic framework through which the next agent state is estimated. However, while in the basic framework this step was as easy as a mapping from a state to another, we need a more complex procedure in our extended framework. Given the current trajectory-set $Tr^i[1, l]$ and the extended model $\mathcal{M}(a_l^i)$, the estimation step is defined in Relational terms as follows:

$$Tr^i[1, l+1] = Tr^i[1, l] \otimes \mathcal{M}(a_l^i) = Tr^i[1, l] \text{ JOIN } (\Delta(a_l^i) \cup \Gamma(a_l^i)). \tag{7}$$

The new trajectory-set $Tr^i[1, l+1]$ is built with the contribution of both Δ and Γ relations. Both relations are in fact used to estimate how the execution of action a_l^i changes the state of the system. Relation Δ is applied to that portion of $\mathcal{B}_l^i$ where action a_l^i is fully enabled. Whereas, relation Γ is applied to all those states in $\mathcal{B}_l^i$ where action a_l^i is not enabled; i.e., the occurrence of an exogenous event has already been assumed.

4.3 Refining the Trajectory-Set with Observations

In the basic framework we have assumed that, whenever action a_l^i is completed, the agent receives observation obs_{l+1}^i just about the new agent state S_{l+1}^i. In the extended framework, agent i can also receive observation obs_k^i referring to a past execution step k (i.e., $1 \le k \le l$). In the next section we present the cooperative monitoring protocol that is at the basis of such a message passing among the agents. In this section we discuss how an observation about the past is handled by a given agent i. Intuitively, consuming observation obs_k^i means selecting from $\mathcal{B}_k^i$ all the states that are consistent with it; in Relational terms:

$$\textit{refined}\mathcal{B}_k^i = \text{SELECT}_{obs_k^i} \mathcal{B}_k^i \tag{8}$$

The result is a refined belief state which is less ambiguous than the original one as a number of states inconsistent with the observations have been pruned off.

It is important to note that the *unknown* value is consistent with any "concrete" observed value. Therefore, for each state $s \in \mathcal{B}_k^i$, if a variable v is *unknown* in s, but v is mentioned in obs_k^i, then v assumes the observed value $obs_k^i(v)$ in $\textit{refined}\mathcal{B}_k^i$. Note that we do not allow an observed variable in obs_k^i to assume the value *unknown*.

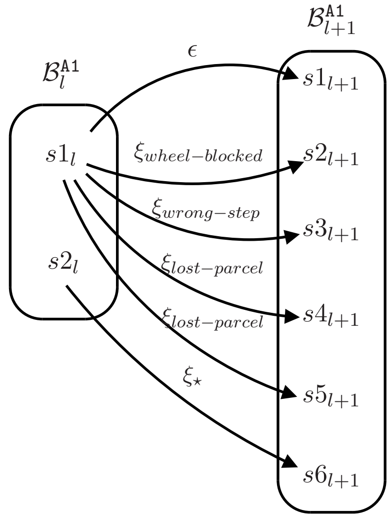

Figure 4: A one-step trajectory-set corresponding to the transition from step l to step $l+1$.

Example 3. Let us consider again the office domain, and assume that after l steps, the trajectory frontier of agent `A1` consists of the following belief state $\mathcal{B}^{\mathtt{A1}}_{l}$:

$$s1_l : \langle\ \texttt{pos = desk1},\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$
$$s2_l : \langle\ \texttt{pos = }\ \mathit{unknown}\ ,\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$

Namely, $\mathcal{B}^{\mathtt{A1}}_{l}$ consists of two alternative agent states $s1_l$ and $s2_l$. Let us assume that the next l-th action performed by `A1` is a `carry` action, whose model has been previously presented in Table 1. According to equation (7), it is easy to see that $s1_l$ matches with state transitions $t1$ through $t5$ of the carry action model (Δ portion of the model), whereas state $s2_l$ matches with transition $t6$ (Γ portion of the model). Figure 4 gives an idea of how these two relations are used to infer the new frontier:

$$s1_{l+1} : \langle\ \texttt{pos = desk2},\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$
$$s2_{l+1} : \langle\ \texttt{pos = desk1},\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$
$$s3_{l+1} : \langle\ \texttt{pos = }\ \mathit{unknown},\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$
$$s4_{l+1} : \langle\ \texttt{pos = desk2},\ \texttt{cObj = }\ \mathit{empty},\ \texttt{Parc2pos}_1\texttt{= desk1}\ \rangle$$
$$s5_{l+1} : \langle\ \texttt{pos = desk2},\ \texttt{cObj = }\ \mathit{empty},\ \texttt{Parc2pos}_1\texttt{=}\ \mathit{unknown}\ \rangle$$
$$s6_{l+1} : \langle\ \texttt{pos = }\ \mathit{unknown}\ ,\ \texttt{cObj =}\mathit{unknown},\ \texttt{Parc2pos}_1\texttt{=}\mathit{unknown}\ \rangle$$

Now, let us assume that agent `A1` receives the observation $obs^{\mathtt{A1}}_{l+1} = \{\langle\texttt{pos = desk2}\rangle\}$, which is used to refine the new frontier. It is easy to see that $obs^{\mathtt{A1}}_{l+1}$ is consistent with all the states except $s2_{l+1}$, in which `pos` is assigned to a different value. States $s3_{l+1}$ and $s6_{l+1}$ are consistent with $obs^{\mathtt{A1}}_{l+1}$ because the *unknown* value is consistent with any precise value. The new refined frontier is therefore

$$s1_{l+1} : \langle\ \texttt{pos = desk2},\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$
$$s3_{l+1} : \langle\ \texttt{pos = desk2}\ ,\ \texttt{cObj = Parc2},\ \texttt{Parc2pos}_1\texttt{= A1}\ \rangle$$

$$s4_{l+1} : \langle\ \texttt{pos = desk2, cObj =}\ \ empty, \texttt{Parc2pos}_1\texttt{= desk1}\ \rangle$$
$$s5_{l+1} : \langle\ \texttt{pos = desk2, cObj =}\ \ empty, \texttt{Parc2pos}_1\texttt{=}\ unknown\ \rangle$$
$$s6_{l+1} : \langle\ \texttt{pos = desk2 , cObj =} unknown, \texttt{Parc2pos}_1\texttt{=} unknown\ \rangle$$

It seems that $s1_{l+1}$ and $s3_{l+1}$ are now identical, indeed we do not just consider single belief states, but trajectories; these two states differ in the way they are achieved: $s1_{l+1}$ is inferred assuming that everything goes smoothly; $s3_{l+1}$ is inferred assuming that something wrong has occurred (i.e., $\xi_{wrong-step}$). Of course, this second hypothesis is not plausible and we will discuss in the next section how it can be pruned off the trajectory-set. ■

This example shows how consuming a set of observations obs^i_k reduces the ambiguity within the agent belief state $\mathcal{B}^i_k$. In addition, the consumption of messages has also a beneficial effect in reducing the ambiguity of the trajectory-set $Tr^i[1,l]$. In fact, the refined belief state can in turn be used to filter the trajectory-set as follows:

$$refinedTr^i[1,l] = \text{SELECT}_{refined\mathcal{B}^i_k} Tr^i[1,l]. \tag{9}$$

The $refinedTr^i[1,l]$ maintains all and only the trajectories that at their k-th step have a state in $refined\mathcal{B}^i_k$. This is an important result since an agent can take advantage of the observations whenever they are available, even though they refer to a past execution step.

It may happen, in fact, that even though obs^i_k is not enough to determine the outcome of action a^i_{k-1}, another belief state $\mathcal{B}^i_h \in refinedTr^i[1,l]$ becomes sufficiently precise to determine the outcome of the pending action a^i_{h-1}. In the next section, we exploit this characteristic to determine the outcomes of pending actions.

4.4 Inferring and Propagating Action Outcomes

Whenever the current trajectory-set is refined with observations, it is useful to scan the pending action list $pActs^i$, and assess, for each action $a^i_k \in pActs^i$, whether either Definition 3 or 4 applies.

The outcome of an action is an important piece of information that we can exploit, as well as observations, to refine the current trajectory-set. The outcome of action a^i_k, either positive or negative, can in fact be used to infer the outcome of other actions in $pActs^i$. To reach this result we exploit the notions of *causal predecessors* of a^i_k ($predecessors(a^i_k)$), and of *causal successors* of a^i_k ($successors(a^i_h)$). First of all, we say that action a^i_h *indirectly provides* action a^i_k with a service, or that a^i_k *indirectly receives* a service from a^i_h, iff there exists a sequence of actions $a^i_{v_1}, \ldots, a^i_{v_n}$ such that:

1. $a^i_{v_1}$ coincides with a^i_h
2. $a^i_{v_n}$ coincides with a^i_k
3. for each action $a^i_{v_x}$, $x : 1..n-1$, there exists a causal link $\langle a^i_{v_x}, q, a^i_{v_{x+1}}\rangle$ in C^i_{local}.

In other words, there must exist a chain of causal links that starts from a^i_h, passes through the actions in the sequence $a^i_{v_1}, \ldots, a^i_{v_n}$, and ends in a^i_k. Indirect causal dependencies that pass through the plans of other agents are not considered by our definition. For example, having the two causal links $\langle a^i_h, q, a^j_v\rangle$ and $\langle a^j_v, q', a^i_k\rangle$, we cannot say that a^i_h

indirectly provides a^i_k with a service since action a^j_v belongs to agent j. This is not a limitation, but an advantage as otherwise the agents should interact heavily in order to compute indirect causal relations. This notion of indirect dependency between actions is at the basis of a locality principle that allows an agent to consider just a portion of its local plan during monitoring and diagnosis.

The set $predecessors(a^i_k)$ is therefore the subset of A^i including all and only the actions that directly or indirectly provide a^i_k with a service. On the other side, $successors(a^i_k)$ is the subset of A^i including all and only the actions which, directly or indirectly, receive a service from a^i_k.

Given an action a^i_k, we denote as $chains\_to(a^i_k)$ the subset of causal links in C^i_{local} defined between actions in $predecessors(a^i_k)$. Similarly, we denote as $chains\_from(a^i_k)$ the subset of causal links in C^i_{local} defined between actions in $successors(a^i_k)$.

Proposition 3 *Let a^i_k be an action whose outcome is* ok*, then all the causal links in $chains\_to(a^i_k)$ represent services that have been satisfied.*

In fact, if a^i_k has outcome *ok*, then all the services required by a^i_k were satisfied, and recursively, all the services required by the actions in $predecessors(a^i_k)$ were satisfied too.

Proposition 4 (Backward Propagation of Success) *Let a^i_k be an action whose outcome is* ok*, and let us mark as* satisfied *all causal links in $chains\_to(a^i_k)$, then any action $a \in pActs^i \cap predecessors(a^i_k)$ having all outgoing links marked as* satisfied*, has outcome* ok*, too.*

Proposition 5 *Let a^i_k be an action whose outcome is* failed*, then the services in $chains\_from(a^i_k)$ might be missing.*

In fact, since a^i_k has outcome $failed$, at least one of its expected effects is missing; on the other hand, action a^i_k could have reached a subset of its effects, and such services could be sufficient to enable some subsequent actions. The forward propagation of the failure must therefore take into account the results successfully achieved. Let us denote as $miss(a^i_k)$ the set of causal links leaving from a^i_k representing missing services.

Proposition 6 (Forward Propagation of Failure) *Let a^i_k be an action whose outcome is* failed*, and let us mark as* missing *each causal link cl in $chains\_from(a^i_k)$ that is reachable from one of the links in $miss(a^i_k)$ via a chain of* missing *causal links, unless cl has already been marked as* satisfied*. Then, any action $a \in pActs^i \cap successors(a^i_k)$, having at least one outgoing link marked as* missing*, has outcome* failed*, too.*

Intuitively, properties 4 and 6 assume that an action performed when it is not fully enabled does not produce correct results. On the other hand, an action that achieves all its effects must have been performed when it was fully enabled, and hence all the services mentioned in its premises must have been provided.

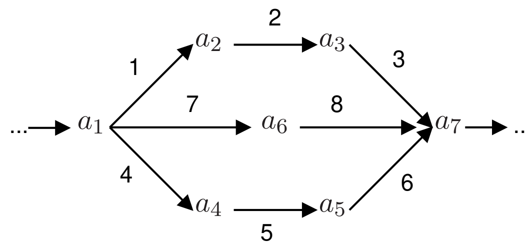

Figure 5: A portion of a local plan restricted to causal links in C^i_{local}.

Example 4. In this example we show how the outcome of an action a is actually exploited to determine the outcomes of other actions. Of course, an agent is able to determine the outcome of a relying on observations and messages from other agents. The cooperative protocol is discussed in details in the following subsection; for the time being, it is important to observe that:

- all the "positive messages" (formalized as *confirm* messages in the following) received at a given step are processed before any "negative message" (i.e., *disconfirm* message) received at the same step;
- an agent receiving at least one "negative message" will stop the execution of its plan, and start a diagnostic phase.

Let us consider the plan segment in Figure 5, where only the links in C^i_{local} are shown, and let us assume that agent i performs these actions in the order a_1, a_2, a_3, a_4, a_5, and a_6, and that all these actions are pending. After the execution of a_5, agent i discovers that a_5 has outcome *ok*. The outcome is propagated backwards: $predecessors(a_5) = \{a_1, a_4\}$, thereby the links in $chains\_to(a_5) = \{4, 5\}$ are marked as *satisfied*; of course, link 6 is also marked *satisfied* because of the nominal outcome of a_5. This enables i to conclude that action a_4 has outcome *ok*; whereas nothing can be concluded about action a_1 since links 1 and 7 are neither marked as *satisfied*, nor as *missing*. Let us assume now that i receives some observations about the service on link 1, and as a consequence it concludes that a_1 has outcome *failed*. In this case the outcome is propagated forwardly: $successors(a_1) = \{a_2, a_3, a_4, a_5, a_6, a_7\}$, thereby $chains\_from(a_1) = \{1, 2, 3, 4, 5, 6, 7, 8\}$; however, links 4, 5, and 6 have already been marked as *satisfied*; in addition, links 7 and 8 are not reachable via chain of *missing* causal links from link 1; thus, only links 2, and 3 are marked as *missing*. Agent i hence concludes that actions a_2 and a_3 have both outcome $failed$. No outcome is inferred for action a_6, which remains *pending*, and no outcome is inferred for action a_7 that has not been performed yet. The outcome of the pending action a_6 is inferred by means of diagnosis inferences discussed in Section 5. ■

Relying on properties 4 and 6, we can determine the outcome of other pending actions by just exploiting the causal dependencies existing among the actions, even though the current trajectory-set is still too ambiguous to apply either Definition 3 (outcome *ok*) or Definition 4 (outcome *failed*).

Note that when we discover that action a_h^i has outcome ok, the exogenous event occurred during that action is necessarily ϵ; thus, we can also filter $Tr^i[1,l]$ as follows:

$$refinedTr^i[1,l] = \text{SELECT}_{e_h=\epsilon} Tr^i[1,l] \quad (10)$$

where e_h refers to the h-th exogenous event labeling the transition from state s_h to state s_{h+1} in $Tr^i[1,l]$. Through the refinement in (10) we keep in $Tr^i[1,l]$ all the trajectories which at their h-th exogenous event have ϵ. Thus we keep the transitions that are obtained through relation Δ, and prune off spurious trajectories contributed by relation Γ.

On the other hand, when the outcome of an action a_h^i is $failed$, we cannot refine the trajectory-set via e_h since we just know that e_h cannot be ϵ, but this is already implicitly obtained thanks to the refinement in equation (9).

Summing up, our weak-committed methodology is able to deal with very scarce observations by using two essential mechanisms. First, we build a trajectory-set maintaining the history of an agent state, and we keep a list of pending action outcomes. Second, we take advantage from observations whenever they are available by revising the knowledge an agent has about itself; in the very favorable case, this revision process can empty the set of pending actions.

4.5 Cooperative Monitoring Protocol

The last element of our CWCM methodology is a cooperative monitoring protocol that allows each agent to exploit information provided by others. The idea is that an agent can take advantage not only of its own direct observations, but also of the observations that other agents have about the environment, and in particular about the shared resources.

The cooperative protocol plays a central role in preserving the resource safeness requirement even when actions that are not fully enabled are performed.

4.5.1 Interaction Scenarios

As in the BaDE strategy, in CWCM two agents, i and j, need to interact with each other when they share a causal link $lk : \langle a_l^i, v = d, a_m^j \rangle$ where $v \in RES$, $d \in dom(v)$, and $v = d$ is a value assignment representing the change in the state of some resource requested by agent j. Contextually to lk, agent i plays the *service provider* role (with $lk \in C_{out}^i$); whereas, agent j plays the role of *service client* (with $lk \in C_{in}^j$). In the following we first present the three interaction scenarios of CWCM. For each of them we shortly report the messages exchanged between the two agents. Then, we present the *client* and *provider* roles in detail by means of high-level algorithms.

- *Notify-ready interaction* In this interaction the provider is sure of having provided the client with the requested service. Thus, the *provider* i sends a message $\langle$about lk notify $v = d$ ready$\rangle$ to the *client* j; no answer from the *client* to the *provider* is required.
- *Notify-not-accomplished interaction* In this scenario, agent i is sure that the requested service is missing; it therefore sends agent j a message $\langle$about lk notify $v = d$ not-accomplished$\rangle$ to *client* j; no answer from j is foreseen.
- *Ask-if interaction* In this case, the *provider* i is unable to determine whether the service $v = d$ has been achieved; thus i asks j for more info by sending j a message

cooperative-protocol::client(inbox, $Tr^i[1, l]$, a^i_l*)*

1. **for each** message m: ⟨ about lk notify $v = d$ ready ⟩ in *inbox* s.t. lk is an incoming link for a^i_l **do**
2. remove m from *inbox*
3. assert $v = d$ in the frontier of $Tr^i[1, l]$
4. **end for**
5. **for each** message m: ⟨ about lk ask-if $v = d$ accomplished? ⟩ in *inbox* s.t. lk is an incoming link for a^i_l **do**
6. remove m from *inbox*
7. **if** unable to observe v **then**
8. reply ⟨ about lk no-info ⟩
9. **else**
10. *obs* ← *observe* v
11. **if** *obs* equals d **then**
12. reply ⟨ about lk confirm $v = d$ ⟩
13. **else if** *obs* is not equal d **then**
14. reply ⟨ about lk disconfirm $v = d$ ⟩
15. **end if**
16. **end if**
17. **end for**
18. **for each** message m:⟨about lk notify $v = d$ not-accomplished⟩ in *inbox* s.t. lk is an incoming message for a^i_l **do**
19. remove m from *inbox*
20. stop plan execution
21. **end for**

Figure 6: The pseudo-code of the cooperative protocol, client behavior.

⟨about lk ask-if $v = d$ accomplished?⟩. The *client* can reply to this message in three different ways:

1. ⟨about lk confirm $v = d$⟩, this message confirms to the *provider* that the expected service $v = d$ has actually been achieved;
2. ⟨about lk disconfirm $v = d$⟩ when the expected service is missing;
3. ⟨about lk no-info⟩ when the *client* is unable to determine whether the assignment $v = d$ holds in the environment or not.

In case i receives a no-info message from j, i will eventually reply either with a ready message or with a not-accomplished one.

4.5.2 Client Role

The algorithm in Figure 6 outlines the behavior of agent i when behaving as a client. This algorithm takes as inputs the *inbox* (i.e., a collector of messages coming from other agents), the current trajectory-set $Tr^i[1, l]$, and the next action to be performed a^i_l.

Agent i consumes a message m from *inbox* only when m is about a service required as a premise for the execution of a^i_l. For each incoming message m of type ready (lines 1 through 4), agent i uses the information provided by another agent as an observation, we use the term "assert" (line 3) as a shortcut for the relational operations presented in equations (8) and (9).

For each incoming message of the *ask-if* interaction (lines 5 through 17), agent i determines whether it is capable of observing v (e.g., is i equipped with the right sensor for v?).

In case i cannot observe v, it replies to the provider with a no-info message. Otherwise, the agent acquires an observation of v, and replies to the provider accordingly.

Finally, whenever agent i receives a not-accomplished message (lines 18 through 21), i just stops the execution of its plan as a service required for performing a_l^i is missing.[1]

It is important to note that an agent playing as a *client* consumes a message m only if m is relevant for the next action to be performed. Thereby, an ask-if message could be answered with a certain amount of delay.

4.5.3 Provider Role

The provider behavior is outlined in Figure 7. The algorithm takes as inputs the $inbox$, the current trajectory-set $Tr^i[1,l]$, the list of pending actions $pActs^i$, and the last performed action a_l^i. More precisely, the last argument can either be $null$, when no action has been performed recently, or an actual action instance whose outcome has still to be assessed. We refine the concept of recently performed action in the next section where we present the main CWCM plan execution loop.

The algorithm starts by checking the $inbox$ in order to consume answers (if any) to previous *ask-if* interactions. The algorithm specifies the behavior of agent i according to the type of received message. In case of confirm messages (lines 1 through 5), agent i uses $v = d$ as an observation to refine its trajectory-set. The term "assert" is used again as a shortcut for the relational operations in equations (8) and (9); the belief state which is actually refined is the k-th +1; that is, the one that contains the effects of action a_k^i. In case of a disconfirm message (lines 6 through 10), agent i prunes off from the k-th +1 belief state in $Tr^i[1,l]$ each state s in which $v = d$ holds. In case of an incoming message of type no-info (line 11 through 18), agent i checks whether all the outgoing links of action a_k^i in C_{out}^i have been marked as ans-no-info, meaning that none of the services provided by a_k^i to other agents have been achieved for sure. If this is the case, agent i marks a_k^i as not-enough-info.

After these preliminary steps, agent i has possibly acquired some further information from others. Thus, it can assess the outcome of all pending actions in $pActs^i$, including a_l^i if not $null$ (line 19). The algorithm in Figure 8 outlines the steps for assessing the outcomes of actions in $pActs^i$, and is discussed later on. Here it is sufficient to say that *assess-pending-actions* returns two lists of actions, *ok-list* and $failed$-*list*, which can be empty, and contain actions whose outcome is *ok* or $failed$, respectively. Of course, whenever an action in $pActs^i$ is found to be either *ok* or $failed$, it is removed from $pActs^i$, and added in the corresponding list. This process also involves actions previously marked as not-enough-info.

If action a_l^i is not $null$ (an action has been performed recently), this is the first time that the outcome of a_l^i is assessed. Thus, in case a_l^i has outcome pending (line 20), agent i starts an *ask-if* interaction (lines 21-24) by asking for further information to all agents that requires one of the services produced by a_l^i. Otherwise, a_l^i is $null$ or a_l^i outcome is not *pending*, and the *ask-if* interaction can be skipped.

From line 25 through line 34, agent i just sends ready and not-accomplished messages according to the actions in *ok-list* and $failed$-*list*, respectively. In addition, agent i sends a

1. The impact of an action failure can be estimated by means of a failure propagation mechanism (Micalizio & Torasso, 2007b). For the sake of discussion, we leave the topic out of this paper.

cooperative-protocol::provider(inbox, $Tr^i[1,l]$, *pActs*^{i}, a^i_l*)*

for each message m:⟨about lk confirm $v = d$⟩ in *inbox* **do**
  remove m from *inbox*
  let lk be $\langle a^i_k, v = d, a^j_m \rangle$
  assert $v = d$ in the k-th $+1$ belief state within $Tr^i[1,l]$
end for
for each message m:⟨about lk disconfirm $v = d$⟩ in *inbox* **do**
  remove m from *inbox*
  let lk be $\langle a^i_k, v = d, a^j_m \rangle$
  prune from the k-th $+1$ belief state within $Tr^i[1,l]$ any state s in which $v = d$
end for
for each message m:⟨about lk no-info⟩ in *inbox* **do**
  remove m from *inbox*
  let lk be $\langle a^i_k, v = d, a^j_m \rangle$
  mark lk as ans-no-info
  if all the links outgoing from a^i_k are marked as ans-no-info **then**
    mark a^i_k as not-enough-info
  end if
end for
$\langle ok\text{-}list, failed\text{-}list \rangle \leftarrow$ *assess-pending-actions*(*pActs*^{i}, $Tr^i[1,l]$)
if a^i_l is not *null* and has outcome *pending* **then**
  for each link lk:$\langle a^i_l, v = d, a^j_m \rangle$, $lk \in C^i_{out}$ $(i \neq j)$ **do**
    send to j message m:⟨ ask-if $v = d$ accomplished?⟩
  end for
end if
for each action $a^i_k \in$ *ok-list* **do**
  for each link $lk : \langle a^i_k, v = d, a^j_m \rangle$ **do**
    send to j message m:⟨about lk notify $v = d$ ready⟩
  end for
end for
for each action a^i_k s.t. ($a^i_k \in failed\text{-}list$) **or** ($a^i_k \in pActs^i$ and marked as not-enough-info) **do**
  for each link $lk : \langle a^i_k, v = d, a^j_m \rangle$ **do**
    send to j message m:⟨about lk notify $v = d$ not-accomplished⟩
  end for
end for
return $\langle ok\text{-}list, failed\text{-}list \rangle$

Figure 7: The pseudo-code of the cooperative protocol, provider behavior.

not-accomplished message for each pending action a^i_k marked as not-enough-info. A pending action marked as not-enough-info highlights how scarcely observable the environment is. In fact, neither agent i, nor other agents waiting for services provided by a^i_k, are capable of determine whether at least one of the expected services has been provided or not. To deal with such an ambiguity, agent i prudentially considers the action as "probably failed". Although this choice could seem strong, it is necessary to preserve the resource safeness requirement in very scarcely observable environments. Agent i has no evidence supporting the successful achievement of the effects expected by a^i_l, and hence i cannot notify the success. At the same time, other agents might be waiting for the services provided by a^i_l, thus these agents would be stalling without even knowing it. Considering a^i_l as failed allows i to get out of the impasse by notifying the failure to the other agents, which may attempt some form of plan repair.

assess-pending-actions($pActs^i$, $Tr^i[1,l]$)
1. $ok\text{-}list \leftarrow \{\}$
2. $failed\text{-}list \leftarrow \{\}$
3. **for each** action $a^i_k \in pActs^i$ **do**
4.   **if** $\forall s \in \mathcal{B}^i_{k+1}, s \models$ **effects**(a^i_k) **then**
5. $ok\text{-}list \leftarrow ok\text{-}list \cup \{a^i_k\}$
6. $Tr^i[1,l] \leftarrow \text{SELECT}_{e_k=\epsilon} Tr^i[1,l]$
7. $oks \leftarrow$ *propagateSuccess*($pActs^i$, a^i_k)
8. $ok\text{-}list \leftarrow ok\text{-}list \cup oks$
9. $pActs^i \leftarrow pActs^i \setminus oks$
10.   **else if** $\forall s \in \mathcal{B}^i_{k+1}, s \not\models$ **effects**(a^i_k) **then**
11. $failed\text{-}list \leftarrow failed\text{-}list \cup \{a^i_k\}$
12. $faulty \leftarrow$ *propagateFailure*($pActs^i$, a^i_k)
13. $failed\text{-}list \leftarrow failed\text{-}list \cup faulty$
14. $pActs^i \leftarrow pActs^i \setminus faulty$
15.   **end if**
16. **end for**
17. remove, if present, mark not-enough-info from any action in *ok-list* or in *failed-list*
18. **return** $\langle ok\text{-}list, failed\text{-}list \rangle$

Figure 8: The pseudo-code for the assessment of the pending actions.

The algorithm terminates by returning the two lists *ok-list* and *failed-list* to the calling algorithm, shown in Figure 10 and discussed in the next section.

4.5.4 Assessing Action Outcomes

Before presenting the main CWCM algorithm, we shortly present the algorithm for assessing the pending actions in $pActs^i$ at a given execution step. As discussed earlier, the assessment relies on properties 4 and 6, and on equation (10). The algorithm is shown in Figure 8; it takes as inputs the list of pending actions $pActs^i$, and the current trajectory-set $Tr^i[1,l]$. The algorithm returns two lists, *ok-list* and *failed-list*, of actions whose outcomes are either *ok* or *failed*, respectively.

The algorithm considers the actions in $pActs^i$ (if any), and for each of them tests whether the action has outcome *ok* or *failed*. In the first case, the success is backward propagated (line 7): *oks* is the list of successfully completed actions discovered by means of the propagation; these actions are removed from $pActs^i$ and added to *ok-list*. In the second case, the failure is forward propagated (line 12): *faulty* is the list of faulty actions discovered by means of the failure propagation; these actions are removed from $pActs^i$ and added to *failed-list*. The algorithm terminates by returning the two, possibly empty, lists *ok-list* and *failed-list*. As mentioned above, an action $a^i_k \in pActs^i$, previously marked as not-enough-info, can be found with a definitive outcome (*ok* or *failed*). This may happen because, although the other agents have not provided i with information about the effects of a^i_k, agent i could exploit the outcome propagation of the actions preceding and following a^i_k. Of course, mark not-enough-info is removed from actions in *ok-list* or in *failed-list*.

Proposition 7 (Protocol Correctness - Resource Usage) *The cooperative monitoring protocol guarantees that the resource safeness requirement is never violated during the* *execution of MAP P. In other words, shared resources are used correctly throughout the plan execution even when action failures occur.*

Proof: Let us consider the interaction scenarios, and show that in each of them the resources are accessed consistently; namely, it never happens that two (or more) agents access the same resource simultaneously.

Given the causal link $lk : \langle a_k^i, res = available, a_m^j \rangle$, the interaction activated by agent i depends on the outcome of action a_k^i.

The *notify-ready interaction* is equivalent to the only interaction of the BaDE framework, and occurs when a_k^i has outcome *ok*. In this case all the expected services have been achieved for sure. Thus, when i notifies j that res is now available, i has already released res: the resource is passed from i to j consistently.

The *ask-if* scenario occurs when a_k^i has outcome *pending*, and splits into three cases.

1. Agent j (i.e., the *client*) directly observes that the resource is available. It can therefore access the resource safely in mutual exclusion.
2. Agent j directly observes that the resource is still occupied by i. In this case j does not attempt to access res. The resource is being used by a single agent and the resource safeness requirement is not violated.
3. Agent j is unable to say whether resource res is available. From the point of view of j, the state of res is *unknown*, and hence, since the preconditions of a_m^j are not satisfied, j keeps waiting for more information from i. Also in this case res is used at most by one agent.

The last interaction scenario occurs when a_k^i has outcome $failed$. In this case, i notifies j that the resource is not available: j does not try to use res as the preconditions of a_m^j are not satisfied. □

Proposition 8 (Protocol Correctness - Provided Services) *Let i and j be two agents playing the roles of provider and client, respectively, about a given causal link $lk : \langle a_k^i, v = d, a_m^j \rangle$. The cooperative monitoring protocol enables the two agents to determine the actual value of variable v or at least to determine whether v is different from the expected value d.*

Proof: The proposition can be proved by considering the different interaction scenarios of the protocol. The *notify-ready interaction* occurs when agent i can conclude that action a_k^i has outcome *ok*. (This may happen by means of direct observations about the effects of a_k^i, or by means of the backward propagation of nominal outcomes.) Since action a_k^i has outcome *ok*, all its effects, including $v = d$, have been achieved. The *ask-if interaction* occurs when agent i cannot determine the outcome of a_k^i, and hence the truth value of statement $v = d$ is not known. In that case agent j is in charge of determining whether the statement $v = d$ is true or false. Agent j can reach this result by means of direct observations on v. The possible answers of j are three:

- j directly observes $v = d$, thus the service has been provided;
- j directly observes that v is not d, thus the service has not been provided;

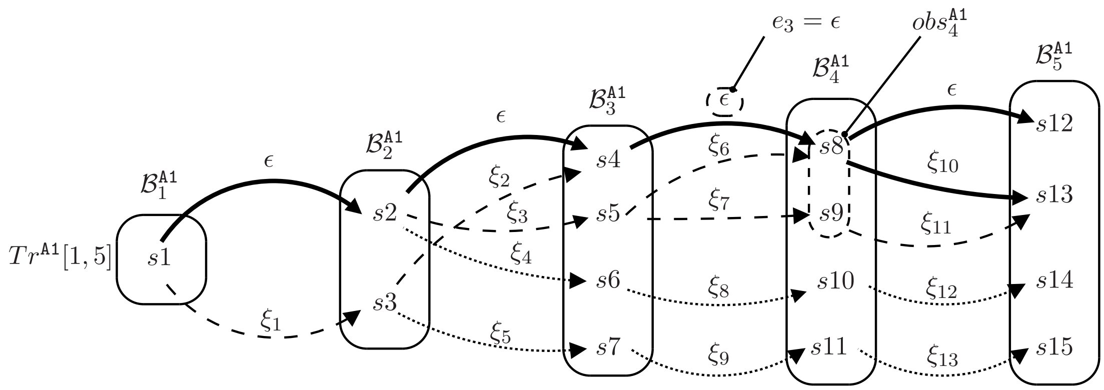

Figure 9: The trajectory-set kept by agent `A1` after the execution of the first four actions.

- j is unable to observe v, in that case agent i relies on answers provided by other agents, if any, asked about the same link. Agent i can only conclude that $v = d$ is true when at least one of the received answers allows it to conclude the nominal outcome of a_k^i; otherwise, the action is assumed failed, and hence also the service $v = d$ is considered as missing.

In the last interaction scenario, agent i has directly observed, or indirectly inferred by means of the forward failure propagation, that $v = d$ is false. □

This proposition can be considered as a sort of generalization of Proposition 7 as it applies to all possible services, not only those services mentioning the *available* value. This proposition is important because allows the agents to diagnose themselves without the necessity of interacting with each other, as we discuss in Section 5.

Proposition 9 (Protocol Complexity) *The number of messages exchanged among the agents is linear in the number n of inter-agent causal links.*

Proof: The *provider-client* interaction occurs only when an agent, playing the role of *provider*, has performed an action a_l^i with at least one outgoing, inter-agent causal link. The number of messages exchanged for handling an inter-agent causal link depends on the outcome of a_l^i. When a_l^i has either outcome *ok* or $failed$, the *provider* sends just one message to the *client* (i.e., ready or not-accomplished, respectively). On the other hand, when a_l^i has outcome *pending*, the two agents exchange each other up to three messages: *provider* sends ask-if, *client* answers no-info, and then *provider* either replies ready or not-accomplished. Thus in the worst scenario, the number of messages exchanged among the agents is $3{\cdot}n$, and hence $O(n)$. □

Example 5. Let us assume that after the execution of the first four actions, the trajectory-set kept by agent `A1` is the one depicted in Figure 9. This trajectory-set contains 5 belief states and none of them is sufficiently refined to determine the outcome of an action; thus, all the actions are currently *pending*. The edges from a state to another labeled with ϵ represent the nominal progress of the plan execution; the others instead, labeled with $\xi_1 \ldots \xi_{13}$, model the occurrence of some exogenous event (possibly $\xi_\star$).

To show how an agent can take advantage of the pieces of information provided by others, let us assume that agent A1 receives from another agent an observation $obs_4^{\mathtt{A1}}$ about the effects of action a_3. For instance, let us assume that action a_3 corresponds to a *move* action, and that observation $obs_4^{\mathtt{A1}}$ refers to the position of agent A1 after the execution of a_3. Observation $obs_4^{\mathtt{A1}}$ is therefore used to refine the belief state $\mathcal{B}_4^{\mathtt{A1}}$. In our example, $s8$ and $s9$ are the only states in $\mathcal{B}_4^{\mathtt{A1}}$ that are consistent with the observation. Thanks to this first refinement (see equations (8) and (9)) we are able to prune off all those trajectories that do not pass either through $s8$ or through $s9$; these trajectories are depicted as dotted edges. Dashed edges, on the other hand, are still possible so these trajectories are still kept within the trajectory set.

However, a further refinement of the trajectory-set is possible when we discover that the refined $\mathcal{B}_4^{\mathtt{A1}}$ is now sufficiently precise to determine that action a_3 has outcome *ok*. In fact, the A1's position conveyed by observation $obs_4^{\mathtt{A1}}$ matches with the expected one; this means that event e_3, affecting a_3, has to be ϵ. By pruning the trajectory-set with $e_3 = \epsilon$ (equation (10)), we are in the fortunate case in which $\mathcal{B}_3^{\mathtt{A1}}$ contains just state s_4, where the nominal effects of action a_2 are satisfied, too. By backwards propagating the success of a_3, first on a_2, and then on a_1 we can conclude that the three actions have all outcome *ok*. In fact, after this process, the resulting trajectory-set maintains just the bold, solid edges; whereas all dashed edges have been pruned off. The resulting trajectory-set, however, does not allow us to conclude anything about a_4, which still remains *pending*. ■

4.6 Cooperative Weak-Committed Monitoring: Main Algorithm

The main CWCM algorithm is outlined in Figure 10. Each agent $i \in \mathcal{T}$ follows this algorithm to execute and monitor its own local plan P^i.

After a few initial steps that set up the agent trajectory-set and the set of pending actions, the algorithm iterates over the actions in P^i as far as the next action to be performed coincides with the pseudo-action a_∞^i, meaning that P^i has been completed. (Remind that we assume all actions in P^i providing atoms in *premises*(a_∞^i) have observable effects.) At the beginning of each iteration, the agent interacts with other agents (line 5) by playing the *client* role of the cooperative protocol. At this step, an agent i consumes ready and not-accomplished messages (if any), and acquires information about the resources required to perform its next action a_l^i. In case the agent receives a not-accomplished message, it stops the plan execution as some of the preconditions for a_l^i will never be satisfied. In case an ask-if message is received, the agent establishes whether it is able to observe the required service and answers accordingly (see algorithms in figures 6 and 7).

Once new information has been acquired and asserted within the agent trajectory-set, agent i assesses whether the next action a_l^i is possibly enabled (Definition 2). In the positive case, the action is performed in the real world (line 8). Subsequently, the agent estimates the possible evolutions of a_l^i by exploiting both $\Delta(a_l^i)$ and $\Gamma(a_l^i)$ to extend the current trajectory-set (line 9). After the completion of action a_l^i, the agent's direct observations are gathered in obs_{l+1}^i (line 10) and asserted in the extended trajectory-set (line 11); also in this case "assert" is a shortcut for the relational operations described in equations (8) and (9). Action a_l^i is then temporarily put into the list of the pending actions (line 12). The outcome assessment is in fact postponed as this step regards all the current pending actions,

Cooperative-Weak-Committed-Monitoring(P^i)
1. $l \leftarrow 1$
2. $Tr^i[1,l] \leftarrow I^i$ //The initial belief state is the initial state of agent i
3. $pActs^i \leftarrow \emptyset$
4. **while** $a^i_l \neq a_\infty$ **do**
5. *cooperative-protocol::client*(*inbox*, $Tr^i[1,l]$, a^i_l)
6. *last* $\leftarrow$ *null*
7. **if** a^i_l *is possibly enabled in frontier of* $Tr^i[1,l]$ **then**
8. execute a^i_l
9. $Tr^i[1,l+1] \leftarrow Tr^i[1,l] \otimes \mathcal{M}(a^i_l)$// trajectory-set extension by using $\Delta(a^i_l)$ and $\Gamma(a^i_l)$
10. $obs^i_{l+1} \leftarrow$ *gather direct observations*
11. assert obs^i_{l+1} in the frontier of $Tr^i[1,l+1]$
12. $pActs^i \leftarrow pActs^i \cup \{a^i_l\}$
13. *last* $\leftarrow a^i_l$
14. $l \leftarrow l+1$
15. **end if**
16. $\langle$*ok-list*, *failed-list*$\rangle$ $\leftarrow$*cooperative-protocol::provider*(*inbox*, $Tr^i[1,l]$, $pActs^i$, *last*)
17. **if** *failed-list* $\neq \emptyset$ **or** $\exists a^i_k \in pActs^i$ marked as not-enough-info **then**
18. stop execution
19. diagnose(P^i, $pActs^i$, *ok-list*, *failed-list*, $Tr^i[1,l]$)
20. *switch to safe mode*
21. **end if**
22. **end while**

Figure 10: Cooperative Weak-Committed Monitoring: high-level algorithm.

and it is activated even when no action has been executed. It is important to note that each iteration of the loop does not necessarily corresponds to the execution of an action. As we have seen, the *provider* behavior of the cooperative protocol needs to know whether an action has been "recently" performed or not (i.e., whether an action has been performed in the current iteration). To this purpose we use variable *last*, which is set to *null* at the beginning of each iteration, and set to action a^i_l only when the action is actually performed (line 13). Whenever an action has been performed, the counter l is incremented (line 14); that is, the l-th plan execution step has been completed.

The while loop proceeds with agent i behaving as *provider* (line 16). This step also includes the evaluation of the outcomes of all the actions in the $pActs^i$ list (see algorithm in Figure 7). The *provider* behavior returns two lists, *ok-list* and *failed-list*, maintaining the actions with outcome *ok* and *failed*, respectively; of course, both lists could be empty. As a side effect, $pActs^i$ is modified by removing any action whose outcome is no longer *pending*. When the list *failed-list* is not empty, or at least one action in $pActs^i$ is marked as not-enough-info, the agent stops the plan execution, and starts a diagnostic process (discussed in Section 5), then switches to *safe mode*. An agent in safe mode does not perform actions, but interacts with other agents trying to reduce the impact of its failure. First of all, an agent in safe mode answers any ask-if message with no-info, this prevents the sender from waiting indefinitely for an answer. Moreover, an agent in safe mode releases as many resources as possible by sending appropriate ready messages; this allows other agents to access those resources and proceed with their plans. A detailed discussion of the safe mode is out the scope of this paper, but it can be found in the works by Micalizio and Torasso (2007b) and Micalizio (2013).

In case no failure has been discovered, and all actions performed so far have outcome *ok* (i.e., $pActs^i$ gets empty), the trajectory-set Tr^i can be simplified. In fact, since all the past actions have a nominal outcome, it is no longer required to keep the whole past history since the beginning of the plan execution. Thus, it is safe and convenient to forget the past and keep within the trajectory-set just the frontier. The implementation we used in our experiments adopts this strategy for keeping the size of a trajectory-set manageable. For the sake of discussion, we do not provide further details on this point.

4.7 Cooperative Weak-Committed Monitoring: Correctness

We conclude this section by discussing the correctness of the algorithm in Figure 10.

Theorem 1 *[CWCM Correctness] CWCM assigns action a^i_k outcome:*
- *ok iff the action has not been affected by exogenous events;*
- *$failed$ iff an exogenous event, possibly $\xi_\star$, has affected a^i_k;*
- *alternatively, CWCM marks a pending action a^i_k as* not-enough-info *iff no outcome can be inferred relying on observations from other agents, nor on the outcome propagation technique.*

Proof: *Part 1*: *action a^i_k has outcome ok iff a^i_k is not affected by exogenous events.* In other words, we have to show that a^i_k reaches $\textbf{effects}(a^i_k)$ iff $e_k = \epsilon$ in each trajectory within $Tr^i[1, l]$, where $k : 1..l - 1$.
($\Rightarrow$) By contradiction, let us assume that $\textbf{effects}(a^i_k)$ have been reached, but the nominal trajectory has been pruned off $Tr^i[1, l]$. This can happen during the monitoring process in just two ways: (a) through observations, or (b) through the outcome propagation. Let us consider case (a), and let us suppose that during the monitoring phase agent i receives observations obs^i_{k+1} consistent with $\textbf{effects}(a^i_k)$. As an effect of pruning $Tr^i[1, l]$ with obs^i_{k+1}, the nominal transition $e_k = \epsilon$ is pruned off $Tr^1[1, l]$; this, however, is in contradiction with the definition of extended model $\mathcal{M}(a^i_k)$, in which only the nominal transitions labeled with ϵ lead to the nominal $\textbf{effects}(a^i_k)$. Thus, either obs^i_{k+1} is inconsistent with $\textbf{effects}(a^i_k)$, and hence a^i_k cannot be *ok*, or obs^i_{k+1} is consistent with $\textbf{effects}(a^i_k)$ and e_k is ϵ in all trajectories within $Tr^i[1, l]$.

Let us now consider case (b), the outcome propagation. There are two cases: backward propagation of *ok*, and forward propagation of $failed$. The backward propagation of *ok* possibly assigns the nominal outcome to actions $a^i_k \in pActs^i$; after the propagation, e_k equals ϵ in each trajectory within Tr^i, by definition. The forward propagation of $failed$ possibly assigns the not nominal outcome to some actions in $a^i_k \in pActs^i$; after the propagation e_k is not ϵ in any trajectory in $Tr^i[1, l]$. The two propagations cannot change the outcome of an action which is not in $pActs^i$: if an action has already been assigned an outcome, that outcome cannot be changed anymore. In particular, if a^i_k has outcome *ok*, and $a^i_h \in predecessors(a^i_k)$ is discovered faulty, then the forward propagation of $failed$ cannot prune $e_k = \epsilon$ from $Tr^i[1, l]$. In fact, as discussed in Proposition 6, the forward propagation impacts only the causal links that are neither marked as *satisfied*, nor as *missing*, and that are along a chain of links starting from one of the links in $miss(a^i_h)$. But if a^i_k has been assigned outcome *ok*, then agent i must have received sufficient observations to determine that the premises of a^i_k were satisfied. It follows that the services required by a^i_k have

already been marked as *satisfied*. Thus, the nominal transition $e_k = \epsilon$ cannot be lost as an effect of the outcome propagation.
($\Leftarrow$) If a^i_k is not affected by an exogenous event, and hence $e_h = \epsilon$ in each trajectory within $Tr^1[1,l]$, then a^i_k has outcome *ok* (i.e., reaches **effects**(a^i_k)). By construction, the extended model $\mathcal{M}(a^i_k)$ guarantees that only the transitions labeled as ϵ leads to states where all expected effects hold. It follows that, when $e_k = \epsilon$ in all trajectories in $Tr^i[1,l]$, a^i_k must have outcome *ok* necessarily.
Part 2: *action a^i_k has outcome $failed$ iff an exogenous event, possibly $\xi_\star$, has affected its execution.* This can be demonstrated following a reasoning similar to the one in Part 1; we omit it for brevity.
Part 3: *action a^i_k marked as* not-enough-info *iff no outcome can be inferred relying on observations from other agents, nor on the outcome propagation technique.* It is easy to see that CWCM marks a^i_k as not-enough-info only in one occasion: in the provider behavior, after that all the answers gathered about a^i_k in an *ask-if* interaction are no-info. This exactly means that no other agent in the team can provide information about the services provided by a^i_k. On the other hand, the marking is removed only after that a^i_k has been inserted either into *ok-list* or $failed$-$list$; thus it cannot happen that an action with a definite outcome is also marked as not-enough-info. □

Theorem 2 *Given the MAP system $\mathcal{S} = \langle \mathcal{T}, RES, P\rangle$, where P is the plan $\langle I, G, A, R, C\rangle$, the global goal G is achieved from I iff all actions in A have outcome ok.*

Proof: In the previous theorem we have demonstrated that an action has outcome *ok* only when all its effects have been achieved, and that such an outcome cannot be changed as an effect of further refinements of the trajectory-set. Thereby, if the global goal G has been reached, all actions in A must have reached their effects, and hence must have outcome *ok*. In fact, since we assume that P has no redundant action (i.e., each action in P contributes to G), it is sufficient that at least one action fails reaching one effect to have: 1) at least one action has outcome $failed$, and 2) at least one piece of G has not been achieved.

On the other hand, if all actions in A have outcome *ok*, G must have been achieved necessarily. By absurd, all actions in A are *ok*, but G has not been reached. This can only happen when P has a flaw, and does not produce G even under nominal conditions, against the initial assumptions (see Section 2) that P is flaw-free and actually produces G. □

Example 5 can be used to clarify the proof. In this example we have shown that, when we restrict $\mathcal{B}^{\texttt{A1}}_4$ to a belief state in which each state satisfies the expected effects of action a_3, then action a_3 has outcome *ok*. At the same time, this outcome is backward propagated so that only edges labeled with ϵ can lead to $\mathcal{B}^{\texttt{A1}}_4$. If action a_4 were the last action of `A1`'s plan, the effects of such an action must be observable, by hypothesis. Now, depending on the available observations, agent `A1` can either conclude that $s12$ is the actual state after a_4 (thereby: (1) the goal has been reached, (2) the trajectory-set contains just one trajectory in which each edge is labeled with ϵ, and (3) all actions have outcome *ok*), or $s13$ is the actual agent's state, and hence at least one action (i.e., a_4 itself) must have outcome $failed$.

Corollary 1 *When the global goal G is achieved, each agent $i \in \mathcal{T}$ keeps in its trajectory-set $Tr^i[1,l]$ only the nominal trajectory $\langle s_1, \epsilon, s_2, \ldots, \epsilon, s_{l+1}\rangle$, where $s_1 \models I^i$ and $s_{l+1} \models G^i$.*

Proof: This follows from the two previous theorems. If G has been reached, all actions in A have outcome *ok* (Theorem 2). On the other hand, since a^i_k is *ok* iff $e_k = \epsilon$ in every trajectory within $Tr^i[1,l]$ (Theorem 1), it follows that each agent i only keeps in its trajectory-set the nominal trajectory. □

The correctness of the monitoring process can therefore be summarized in the following statement: When the execution of P is not affected by any anomalous event, the cooperative monitoring is able to keep a trace of the progress until the achievement of the goal G since the nominal transition is never lost. On the other hand, when the execution of P is affected by at least one anomalous event, even not known in advance, the cooperative monitoring is able to detect it and to stop the execution phase. In addition, Proposition 7 assures that in nominal, as well as anomalous, situations the resources are always accessed consistently.

5. Plan Diagnosis: a Local Strategy

Plan diagnostic inferences start as soon as the CWCM algorithm has discovered the failure of at least one action (i.e., $failed$-*list* is not empty), or a pending action is marked as not-enough-info. In this section we discuss what we mean by *plan diagnosis*, and how it can be inferred. We propose a distributed approach in which each agent infers a diagnosis about its local plan autonomously. In fact, thanks to Proposition 8, the plan execution is *safe* with respect to the use of resources, so an agent can never blame other agents to explain its own action failures.

5.1 Inputs from CWCM

In the previous section we have focused on the monitoring purpose of the CWCM methodology. It is important to note, however, that CWCM also produces useful pieces of information from a diagnostic point of view. First of all, the actions in $failed$-*list* could be considered as a plan diagnosis according to the definition by Roos and Witteveen (2009); namely, a subset of actions that when assumed faulty explain the observations. However, in $failed$-*list* we do not take into account that some action failures might be the indirect consequences of others. Thus, $failed$-*list* is not sufficient as we would like to isolate the *primary action failures* that have caused other *secondary action failures*.

In addition, CWCM produces a trajectory-set $Tr^i[1,l]$, which can be seen as a set of consistency-based diagnoses (Reiter, 1987). Each trajectory in $Tr^i[1,l]$ is a possible explanation for the agent's behavior consistent with the observations received by the agent itself.

5.2 Event-Based Explanations

Dealing directly with $Tr^i[1,l]$, however, might be awkward since it encodes all the possible explanations, including the ones mentioning the indefinite exogenous event $\xi_\star$, which should be considered as very unlikely. Moreover, trajectories which share the same sequence of events, but differ for a few state variables, are considered as completely different explanations. Thus, $Tr^i[1,l]$ needs to be processed in order to be useful. A first reduction of $Tr^i[1,l]$ is given by projecting it over the event variables $e_1, \dots, e_{l-1}$; we call the resulting structure Event-based Explanations (EVE):

$$EVE = \text{PROJECT}_{e_1,\ldots,e_{l-1}} Tr^i[1,l]. \tag{11}$$

EVE is a set of sequences of exogenous events (including ϵ and $\xi_\star$). Each sequence in this set is a possible consistency-based diagnosis for the anomalous behavior of the agent. Since EVE could still contain a huge number of diagnoses, EVE is not very informative for a human user who has to decide how to recover from a plan failure. One way for further reducing the number of diagnoses would be to prefer diagnoses which involve the minimum number of exogenous events. Unfortunately, this preference criterion would lead to misleading results because events are dependent on one another. To find meaningful explanations, one should identify what exogenous events have caused primary action failures and what exogenous events correspond to secondary action failures.

5.3 Minimum Primary Action Failures

To facilitate the identification of primary action failures, we distinguish between indefinite events $\xi_\star^\Delta$ contributed by the Δ portion of an action model, and indefinite events $\xi_\star^\Gamma$ contributed by the Γ portion. While this distinction is not necessary in CWCM, it turns out to be useful for the diagnostic purpose. Intuitively, $\xi_\star^\Delta$ denotes the occurrence of an exogenous event affecting the execution of a (possibly) enabled action; $\xi_\star^\Delta$ is therefore an *unknown* abrupt change affecting the nominal behavior of an action. On the other hand, $\xi_\star^\Gamma$ is just the indefinite event we use to label state transitions when an action has been performed from a state not satisfying its preconditions. Relying on this distinction, it is possible to identify a primary failure by means of the following definition.

Definition 7 *An action $a_k \in pActs^i \cup failed\text{-}list$ is a* primary action failure *iff there exists an explanation $x \in EVE$ such that $x[e_k] \neq \epsilon$ and $x[e_k] \neq \xi_\star^\Gamma$, where $x[e_k]$ is the k-th event in explanation x.*

In other words, an action a_k is considered as a primary failure in a given event-based explanation $x \in EVE$ iff the occurrence of an exogenous event mentioned in $\Delta(a_k)$ is assumed in x. Note that in Definition 7 we also examine the set of pending actions $pActs^i$, including actions marked as not-enough-info. In addition, note that the set of primary action failures can never be empty. In fact, an agent starts a diagnosis phase only when one of its performed actions has been labeled as *failed*. On the other hand, when an agent stops the execution of its plan because of another agent fails in providing a service, the first agent is exonerated from diagnosing itself since none of its actions have been labeled as *failed*, and the root causes for the missing service have been located outside its plan.

Secondary failures are caused by a primary failure, and are defined as follows:

Definition 8 *Let $x \in EVE$ be a possible explanation, let $a_k \in failed\text{-}list \cup pActs^i$ be a primary failure in x, then all actions $a_h \in successors(a_k)$ such that $x[e_h] = \xi_\star^\Gamma$ are* secondary failures *caused by a_k according to explanation x.*

Note that, given a primary failure a_k in an explanation $x \in EVE$, not all the actions in $successors(a_k)$ are necessarily secondary failures (see Proposition 4). In fact, even though a_k has not achieved all its effects (i.e., it has outcome *failed*), the action may have reached some of them. As a consequence, some of the actions in $successors(a_k)$ may be enabled despite the failure of a_k. For this reason, in Definition 8 we require that an action $a_h \in successors(a_k)$ is labeled as a secondary failure only when the exogenous event $\xi^{\Gamma}_{\star}$ is assumed in the explanation x. From the definitions of primary and secondary failures the proposition below follows directly.

Proposition 10 *Given an explanation $x \in EVE$, the set of primary action failures Prm_x, and the set of secondary action failures Snd_x extracted from x are disjointed.*

Relying on this proposition, we define *Primary Action Failure Diagnoses* (PADs):

Definition 9 *Let $x \in EVE$ be a possible event-based explanation, the* primary action-failure explanation *(PAD) extracted from x is the pair $\langle Prm_x, Snd_x \rangle$ such that Prm_x and Snd_x are the sets of primary and secondary failures, respectively, extracted from x.*

Of course, since EVE in general contains several explanations, and since primary failures are assumed to be independent of each other, it is possible to extract the minimum cardinality primary action-failure diagnoses (mPADs) by simply selecting the explanations with the minimum set of primary failures:

$$mPADs = \{Prm_x \text{ such that } x \in EVE \text{ and } |Prm_x| \text{ is minimum } \} \tag{12}$$

Minimum primary action failure diagnoses (mPADs) are indeed what we mean for plan diagnosis: they localize which actions should be qualified as failed in order to explain the anomalous observations.[2]

5.4 Refining the Plan Diagnosis

Having inferred plan diagnosis, one can refine these diagnoses by identifying their root causes. Our *refined explanations* are expressed in terms of exogenous events, and can be extracted from the EVE set.

Definition 10 *Let a_h be a primary action failure, and let $EVE(a_h)$ be the set of explanations $x \in EVE$ such that $a_h \in Prm_x$, then the* refined explanation *for action a_h is*

$$refinedExp(a_h) = \bigcup_{x \in EVE(a_h)} x[e_h]. \tag{13}$$

In other words, $refinedExp(a_h)$ consists of all the exogenous events that might have occurred during the execution of action a_h, and hence might have caused the failure of a_h. Of course, since a_h is a primary failure, all secondary failures caused by a_h can also be explained by the occurrence of one of the events in $refinedExp(a_h)$.

2. Note that different preference criteria could be adopted to select explanations in EVE. For instance, one could prefer minimality rather than minimum cardinality.

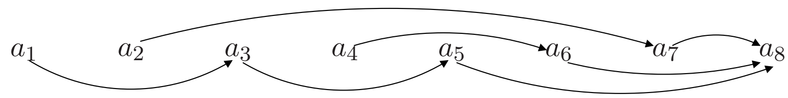

Figure 11: A portion of the local plan assigned to agent i

Example 6. Let us consider the simple local plan in Figure 11 assigned to agent i. To simplify the picture we just show the local causal links in C^i_{local}. Let us assume that, after the execution of such a local plan, agent i detects the failure of action a_8. The diagnostic process is activated in order to explain such a failure by identifying its (minimum) set of primary action failures. The diagnostic process receives in input the list of failed actions $failed\text{-}list=\{a_8\}$, the list of successfully completed actions $ok\text{-}list=\{a_4\}$, and the list of the pending actions $pActs^i = \{a_1, a_2, a_3, a_5, a_6, a_7\}$. In addition, the diagnostic process receives also the trajectory-set $Tr^i[1, 9]$, but for simplicity we show in Table 2 just the set of event-based explanations (EVE) extracted from the trajectory-set.

From Table 2 it is easy to see that all the explanations, except the last one, explain the failure of action a_8 as an indirect effect of a previous failure (i.e., a_8 is a secondary failure). Only the last explanation considers a_8 as a primary failure, but an unknown, and very unlikely, exogenous event $\xi^{\Delta}_{\star}$ must be assumed.

The first step of the diagnostic process consists in inferring the set of $mPADs$ diagnoses. Thus, we identify primary and secondary failures for each explanation in EVE:

$$
\begin{array}{rrll}
PAD : \{ & x1, x2 : & \langle\{a_1\}, & \{a_3, a_5, a_8\}\rangle \\
& x3 : & \langle\{a_6, a_7\}, & \{a_8\}\rangle \\
& x4 : & \langle\{a_3, a_6, a_7\}, & \{a_8\}\rangle \\
& x5 : & \langle\{a_2\}, & \{a_7, a_8\}\rangle \\
& x6 : & \langle\{a_8\}, & \emptyset\rangle \ \}
\end{array}
$$

We can observe some interesting consequences. First of all, some explanations in EVE are collapsed within a single explanation in $PADs$; see for instance explanations $x1$ and $x2$. This is an advantage as we can reduce the number of alternative explanations. In addition, the sets of primary action failures can be used to identify (subset-)minimal diagnoses. For instance, explanation $\{a_6, a_7\}$ derived from $x3$ is a minimal diagnosis, whereas explanation $\{a_3, a_6, a_7\}$ extracted from $x4$ is not. Finally, since we assume that primary failures are independent of each other, we can prefer the subset-minimal diagnoses whose cardinality is

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
$x1$	ξ_1	ϵ	$\xi^{\Gamma}_{\star}$	ϵ	$\xi^{\Gamma}_{\star}$	ϵ	ϵ	$\xi^{\Gamma}_{\star}$
$x2$	ξ_5	ϵ	$\xi^{\Gamma}_{\star}$	ϵ	$\xi^{\Gamma}_{\star}$	ϵ	ϵ	$\xi^{\Gamma}_{\star}$
$x3$	ϵ	ϵ	ϵ	ϵ	ϵ	ξ_2	ξ_3	$\xi^{\Gamma}_{\star}$
$x4$	ϵ	ϵ	ξ_6	ϵ	ϵ	ξ_2	ξ_3	$\xi^{\Gamma}_{\star}$
$x5$	ϵ	ξ_4	ϵ	ϵ	ϵ	ϵ	$\xi^{\Gamma}_{\star}$	$\xi^{\Gamma}_{\star}$
$x6$	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	$\xi^{\Delta}_{\star}$

Table 2: The set EVE maintained within the current trajectory-set

minimal. In our example they are $mPADs = \{\{a_1\}, \{a_2\}, \{a_8\}\}$. In fact, it is thus sufficient to assume the failure of one of these actions to explain the observations.

As a further step, for each action in $mPADs$, one can also infer a refined diagnosis. For instance, it is easy to see that the primary action failure a_1 has two alternative refined diagnoses: either ξ_1 or ξ_5 (see Table 2); whereas the primary action failure a_2 has ξ_4 as single possible refined diagnosis. Finally, one has to assume the occurrence of $\xi_\star^\Delta$ to explain the primary action failure a_8. Relying on refined diagnoses, other preference criteria could be employed and conclude that the primary failure a_8 is less likely than a_1 and a_2, and hence it could be disregarded. ■

Note that, since each agent is able to diagnose its own plan autonomously, a plan diagnosis at global level could be inferred by combining the local solutions inferred by each agent in the team, and such an integration is guaranteed to be globally consistent. In fact, thanks to Proposition 8 an agent can never blame another agent for the failure of one of its actions.

6. Experimental Analysis

So far we have addressed both the CWCM methodology and the diagnostic inferences in a declarative manner by means of relations and Relational operators between relations. Relations are a simple, yet powerful formalism to represent nondeterministic action models and ambiguous belief states. In addition, they can also be used to model very complex structures such as the trajectory-set and the event-based explanations (EVE).

When it comes to actually implementing the CWCM methodology, however, it must be noticed that the computational complexity of the algorithm in Figure 10 is dominated by the complexity of the (macro-)operator $\otimes$ involved in the extension of the current trajectory-set. On the other hand, the diagnostic inferences are based on the projection of the current trajectory-set over the event variables (see equation 11). Both these steps might be computationally very expensive, and an efficient implementation of relations and relational operators therefore becomes essential. A possible way to cope with this issue is to translate the relations into some symbolic, and hence compact, formalism, and then encode the Relational operators as operations in the selected symbolic formalism. Alternatively, it may be possible to exploit the recent advancements in Continuous Query Languages (CQLs) to deal with data streams (see e.g., the STREAM system in Arasu, Babu, & Widom, 2006), and implement CWCM relying on the primitives made available by the Data Stream Management System at hand.

In this paper, we have chosen the method of knowledge compilation, and in particular, we have selected the Ordered Binary Decision Diagram (OBDD) (Bryant, 1986, 1992) formalism to encode relation and Relational operators. This choice is justified by two main reasons: first, OBDDs are nowadays a well-known language made available through many mature libraries; second, the theoretical results by Darwiche and Marquis (2002) suggest that OBDDs can answer most of queries in polynomial time provided that their sizes remain tractable. An in-depth description on how the cooperative monitoring and diagnosis have been implemented via OBDDs is reported in the Appendix.

The rest of the section is organized as follows. First, in Section 6.1, we sketch the software architecture of our implementation; then in Section 6.2, we present the experimen-

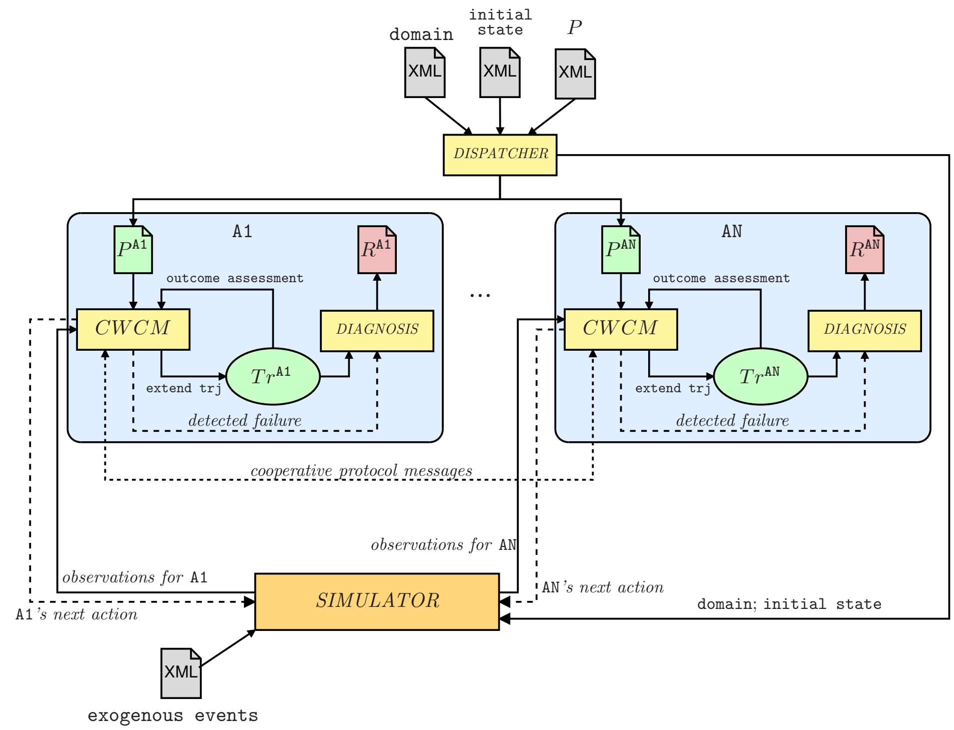

Figure 12: The software architecture of the CWCM implementation used in tests.

tal setting we used to carry out the tests, consisting in the simulated execution of several MAPs. Finally, we discuss the most interesting results about monitoring (Section 6.4), and diagnosis (Section 6.5).

6.1 Software Architecture and Implementation

The CWCM proposal has been implemented as a Java SDK 7 program. The software architecture is shown in Figure 12, highlighting the main actors: the Dispatcher, the N agents of a team (from agent `A1` to agent `AN`), and the Simulator. The picture also shows the internal architecture of the agents. Solid edges between modules represent data flows, dashed edges represents instead control flows, whereas the dotted edge between $CWCM$s abstracts all the messages exchanged by the agents during the cooperative monitoring. The simulation of a MAP P starts by submitting to the Dispatcher module three XML files containing, respectively, the system domain (i.e., what agents and objects are defined in the scenario at hand), the system initial state (e.g., the initial positions of agents, the initial states of resources, etc.), and the MAP P to be performed. The Dispatcher decomposes P into local plans so that each agent will receive just the portion of P of its interest. In particular, once P has been decomposed, the Dispatcher activates the agents, which are implemented as threads, by passing them their initial states and their local plans.

OBDDs are made available through the JavaBDD library [3], which provides a java, easy-to-use interface between Java and BuDDy [4], a popular and mature library for manipulating OBDDs written in C.

Besides the agents, the Dispatcher activates also a Simulator, implemented as a thread. Differently from agents, however, the Simulator does not receives in input a plan, but just the initial state of the system. In addition, the Simulator reads from a fourth XML file the exogenous events that have to be injected during the plan execution. More precisely, the file is a list of agents' actions, each of which is associated with the anomalous event that must occur during the execution of that action. Of course, only the subset of actions to be affected by exogenous events are mentioned in this file.

Once the environment has been set-up, the Dispatcher starts the agents, which will execute the CWCM algorithm as discussed in Section 4. The actual execution of an action is just simulated by the Simulator: Whenever an agent intends to perform an action, it sends a message to the Simulator conveying the action to be performed. The Simulator will simulate the action execution taking into account possible exogenous events that have to be injected. If the action is associated with observations, the Simulator sends to the corresponding agent an appropriate message. It is worth noting that also the Simulator, like any other agent, uses OBDDs to estimate the next state of the whole system according to the actions that are currently in progress. Differently from the agents, however, the Simulator always knows the precise state of each agent and resource in the system. Some more details about the use of OBDDs for handling relations are given in Appendix A. As discussed in Section 4, whenever the failure of an action is detected by an agent i, the Diagnosis module of that agent is activated. The results of the diagnosis inferences, discussed in Section 5 are saved in a report file R^i associated with agent i.

The experiments described in the following were performed on a PC, Intel Core 2 Duo, 2.80 Ghz, 8 GB RAM equipped with Windows 7 OS. Each test is repeated ten times, and average values are considered in the experimental analysis in order to absorb load fluctuations of the CPU.

6.2 Experimental setting

The domain we used for our tests has already been introduced in Example 1. The actions each agent can perform are summarized in Table 3 [5], reporting some details about the encoding of the action models as OBDDs. More precisely, *# variables* is the number of state variables over which the OBDD is defined; this number includes one variable for encoding the possibly anomalous event occurring during the action execution; the remaining variables are used to encode an agent state transition from step t, when the action starts, to step $t+1$, when the action ends. Columns $\#\Delta$*-nodes* and $\#\Delta$*-trans.* report, respectively, the number of nodes of the OBDD encoding the Δ portion of the action model, and the number of state transitions encoded by Δ. Columns $\#\mathcal{M}$*-nodes* and $\#\mathcal{M}$*-trans.* refer to the whole extended model $\mathcal{M}$, including the Γ portion. In such a domain, each agent handles 36 variables to encode its own belief state about the environment.

3. http://javabdd.sourceforge.net/index.html
4. http://sourceforge.net/projects/buddy/
5. Examples of test cases and action models can be found at http://www.di.unito.it/∼micalizi/CWCM/index.html.

	# variables	#Δ-nodes	#Δ-trans.	#$\mathcal{M}$-nodes	#$\mathcal{M}$-trans.
move	15	193	34	291	420
carry	19	346	38	374	1857
load	17	386	40	892	169
unload	17	386	40	892	169

Table 3: Some details on the relational action models.

6.3 Objectives of the Experimental Analysis

There are at least three main questions that we want to get answered by means of our experiments. These questions are:

- Does CWCM scale up well as the number of agents in the team grows?
- Is CWCM affected by the level of system observability? and if so to what extent?
- Is the cooperation among agents really useful for the monitoring purpose?

To answer these questions, we carried out our tests by varying three main characteristics: *team size*, *observability level*, and *monitoring strategy*.

6.3.1 Team Size

To assess the scalability of CWCM, we have generated MAPs with teams from 3 to 8 agents. Thus, we have 6 scenarios, and for each of them, we have synthesized 30 MAPs. The main characteristics of these MAPs are reported in Table 4. Note that the MAPs are not trivial as they consist of a significant number of actions and subgoals to be achieved. The term *MAP-span* refers to the number of execution steps that are required to complete the plan under nominal conditions and full observability. The concurrency rate, computed as the number of actions divided by *MAP-span*, indicates that agents do perform actions concurrently. Finally, the number of inter-agent causal links shows how often agents interact with each other to achieve their own subgoals.

6.3.2 Observability Level

To assess the competence of CWCM, the MAPs were performed under different conditions of observability. In particular, we considered three degrees of domain observability. In the following, the term *FULL* denotes a complete observability of the effects of the actions performed by the agents. Such a level of observability is unrealistic in practice, but it represents our benchmark to compare the performance of CWCM in the other observability conditions. The term *HIGH* denotes a degree of observability that guarantees to observe the effects of 70% of the MAPs actions, randomly selected. Finally, the term *LOW* denotes a degree of observability of just 30% of the MAPs actions, again randomly selected.

6.3.3 Monitoring Strategies

Finally, to assess the actual benefits achieved by the cooperation among the agents during the monitoring phase, we considered three alternative monitoring strategies:

- **BaDE**, already presented in Section 2, is the simplest strategy, based on the strong committed policy.

scenario	#agents	#actions	#subgoals	MAP-span	concurrency rate	#causal links	#inter-agent links
SCN3	3	67.67 ±14.04	47.00 ±10.20	30.78 ±9.94	2.2	226.44 ±32.82	10.5 ±1.87
SCN4	4	69.00 ±5.42	52.00 ±3.15	27.90 ±2.93	2.5	239.30 ±13.39	20.1 ±1.45
SCN5	5	91.60 ±7.37	77.80 ±3.97	34.40 ±3.10	2.7	329.80 ±11.34	26.8 ±1.1
SCN6	6	128.90 ±40.27	73.60 ±4.63	27.00 ±8.10	4.8	377.70 ±70.14	28 ±4.75
SCN7	7	180.40 ±36.84	73.90 ±18.35	30.90 ±5.27	5.9	467.20 ±60.55	36.6 ±4.92
SCN8	8	156.40 ±6.13	48.80 ±7.63	20.50 ±2.70	7.2	360.60 ±6.90	45.00 ±2.82

Table 4: Characteristics of the MAPs in the six scenarios under nominal conditions (average values and confidence intervals).

- **WCM** (Weak-Committed Monitoring) introduced by Micalizio and Torasso (2008, 2009) is based on the weak-committed policy that allows agents to keep trajectory-sets to cope with scarce observability. In WCM, an agent i is able to keep pending actions as far as these actions do not provide services to other agents. Differently from CWCM, in WCM agents cannot cooperate with each other; therefore, when the outcome of an action a cannot be precisely determined, and a provides another agent j (i.e., $i \neq j$) with a service, a is assumed as failed by i, which also stops the execution of its own plan.
- **CWCM**, discussed in Section 4, extends the weak-committed policy with the active cooperation among the agents.

6.3.4 Exogenous Events

Although exogenous events have been generated randomly, their generation reflects the (expected) probability with which a given exogenous event can occur. For instance, a completely unexpected event, encoded by $\xi_{\star}$, is very unlikely to occur, and hence its frequency in our experiments is pretty low. Table 5 shows the probability distribution used to generate exogenous events randomly.

6.4 Experimental Analysis: Monitoring

The experimental analysis of the monitoring task is subdivided into two main parts. In the first one, we assess the three strategies BaDE, WCM, and CWCM, under nominal conditions; that is, when no exogenous event occurs during the simulated execution of MAPs. The goal is to study the impact of the observability degree on the competence

Exogenous event	Probability
blocked-wheel	25 %
wrong-move	10 %
lose-parcel	25 %
slip-parcel	10 %
blocked-arm	25 %
$\xi_\star$	5 %

Table 5: The exogenous events and their frequencies in the experiments.

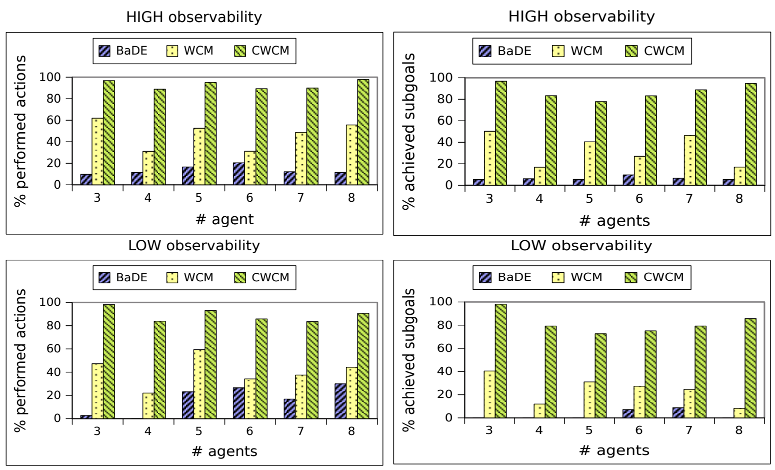

Figure 13: [*Nominal Conditions*] Competence: percentage of performed actions and achieved goals.

of the three strategies. In the second part, we assess again the competence of the three strategies when exogenous events do occur.

6.4.1 Nominal Conditions

Competence. The competence is estimated as the percentage of actions performed and subgoals actually achieved by the agents. Since under the condition of *FULL* observability the agents perform 100% of their actions and achieve 100% of their subgoals in each of the three strategies, in Figure 13 we just report the results under *HIGH* and *LOW* conditions. As expected, BaDE is very sensitive to the observability degree. On the other hand, since WCM and CWCM keep trajectory-sets, they are more tolerant to partial observability, and generally behave much better than BaDE. CWCM does better than WCM as the cooperation between the agents allows them to compensate the lack of direct observations with messages coming from others. As discussed in Section 4.5, however, it may be possible that even the other agents are unable to provide useful pieces of information. Thus, also with the CWCM strategy, an agent decides to stop the execution of its own plan when, even asking other agents for more observations, it is not possible to determine the outcome of an action. As explained in Section 4.5, in this case an agent stops the execution of its plan by marking some actions as not-enough-info. This is the reason why the percentage of performed actions and achieved goals is below 100% with observability levels *HIGH* and

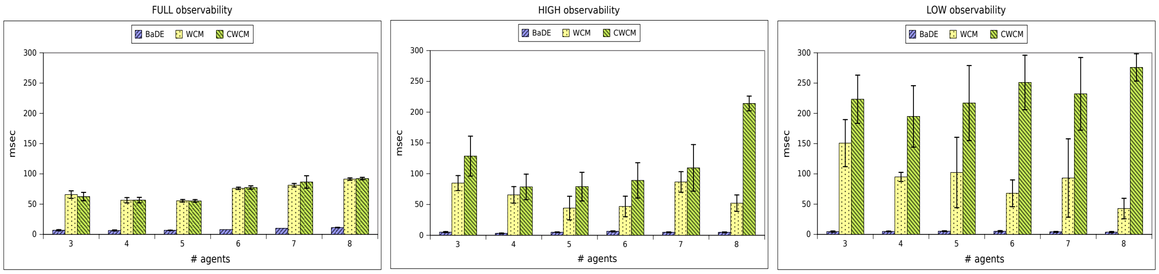

Figure 14: [*Nominal Conditions*] Monitoring time (average and 95% confidence interval) for a single execution step.

LOW. The results obtained by CWCM are in any case remarkable: in the worst case, SCN5, at least 80% of actions have been performed and 70% of subgoals have been achieved despite only 30% of the actions were observable.

Computational Time. Figure 14 shows the average time (and the 95% confidence interval) for monitoring a single step of execution. Note that for the BaDE strategy the monitoring just consists in estimating the next belief state; whereas, WCM and CWCM have to extend their trajectory-sets. In addition, CWCM has also to cooperate with other agents. The cooperation can introduce further costs as the consumption of a message from another agent corresponds to an operation on the OBDD encoding the current trajectory-set. A first positive result emerging from Figure 14 is that, even in the worst scenario, CWCM takes no more than 300 milliseconds for monitoring the execution of an action. This allows us to conclude that CWCM could be employed effectively in real-world domains where agents' actions are performed in the order of seconds.

In addition, it is easy to see that computational time strongly depends on the observability level. For example, under *FULL* observability, CWCM and WCM behave very similarly; in this case, in fact, CWCM agents do not need to cooperate each other, and hence the two strategies are almost the same. However, when the observability decreases, CWCM is slightly more expensive than WCM and BaDE. This higher cost is counterbalanced by the competence of CWCM, that, as already noticed, outperforms the competence of both BaDE and WCM.

From the charts in Figure 14 it is also apparent that there is no strict dependency between the number of agents in the team and the computational time of the three strategies. This, in fact, is a consequence of our distributed approach where each agent maintains its own point of view about the environment, and the cooperation with other agents is just based on the exchange of messages and not belief states.

OBDD dimensions. The relation between time and observability becomes clear when we consider the sizes of the OBDDs encoding the trajectory-sets; see Figure 15, left. For brevity we just report the average sizes of the OBDDs maintained by the three strategies under *HIGH* and *LOW* observability conditions[6]. It is easy to see that there exists a relation between the computational time shown in Figure 14 and the sizes of the OBDDs in Figure

6. In the *FULL* observability case, the OBDD sizes for CWCM are well below 3000 nodes, on average. In addition, CWCM and WCM generate OBDDs with similar sizes, as expected.

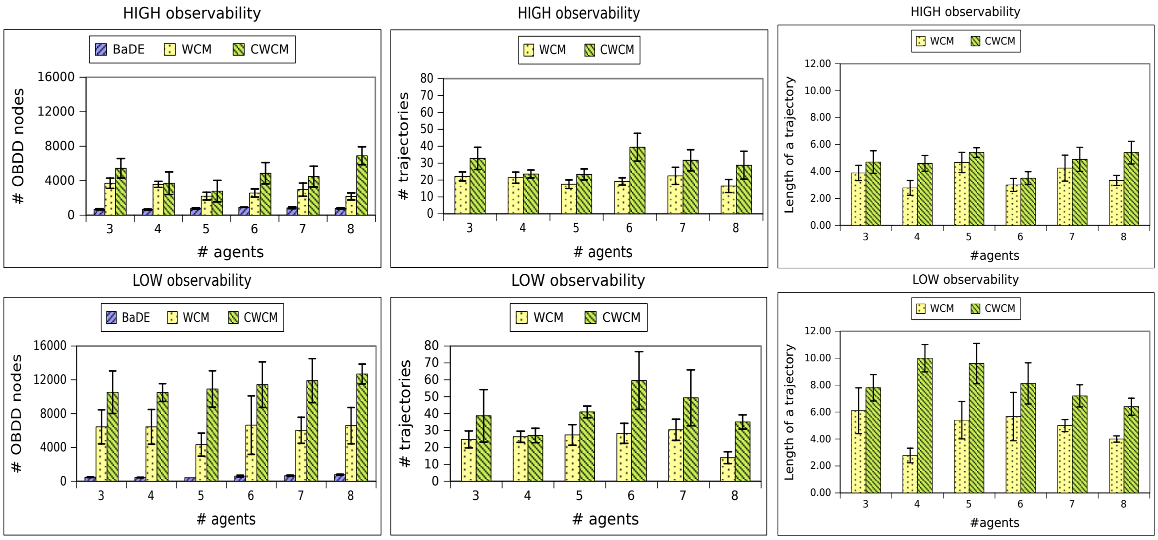

Figure 15: [*Nominal Conditions*] Left: Sizes of OBDDs in number of nodes (average and 95% confidence interval); center: Average number of trajectories within a trajectory-set; right: Average length of one trajectory.

15, left: the bigger the OBDDs the higher the computational time. As we have already noted, although OBDDs may get very large, the computational time is still acceptable. (The biggest OBDD that has been observed had 17,707 nodes, and was built by CWCM in SCN5 under *LOW* observability.)

Obviously, the level of observability has a strong impact on the dimensions of the OBDDs. In fact, a reduced level of observability makes the trajectory-sets more ambiguous, and hence more trajectories have to be encoded within a single OBDD. This is made explicit in Figure 15, center, where we show the number of trajectories encoded, on average, within a trajectory-set at each time instant, and their length (Figure 15, right). Of course, in these two last charts, we only consider WCM and CWCM since the BaDE strategy does not build trajectory-sets. Moreover, note that in the actual implementation of CWCM, the extension of a trajectory-set does not cover the whole plan performed so far, but only the current subset of pending actions.

CWCM Communication Analysis. We conclude the study under nominal conditions with an analysis of the communication required by the CWCM methodology. Figure 16 shows the average number of messages exchanged among the agents. The first interesting result is that, under *FULL* conditions, the number of exchanged messages coincides with the number of inter-agent causal links. In fact, since these results are taken under nominal conditions, each action reaches its nominal effects; therefore, the cooperative protocol handles each inter-agent causal link by means of a simple ready message sent by the *provider* to the *client*, no answer is required. When the observability level is just *HIGH*, however, the number of messages tends to increase, even though it does not increase significantly except for scenario SCN8. As expected, the largest number of messages is exchanged when the observation is *LOW*, as expected.

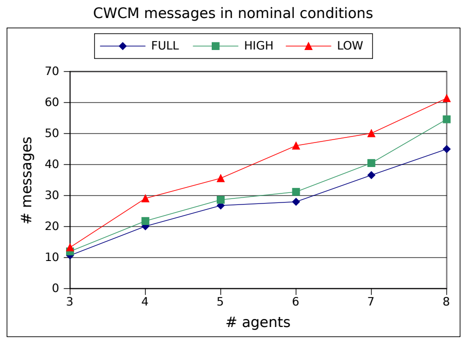

Figure 16: The number of messages exchanged by CWCM agents in nominal conditions.

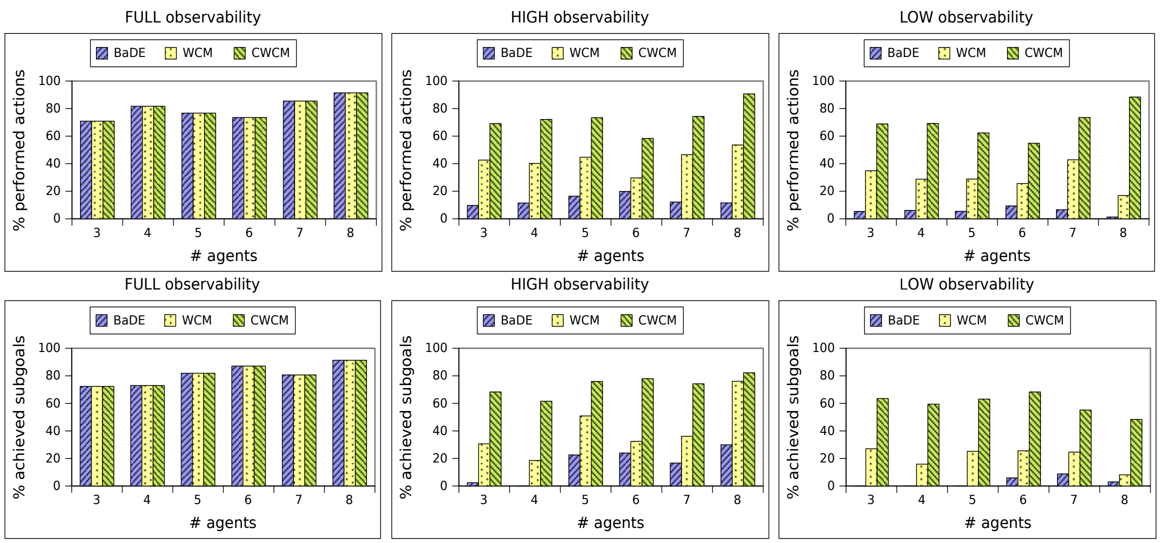

Figure 17: [*Faulty Conditions*] Competence: percentages of performed actions and achieved goals.

6.4.2 Faulty Conditions

Competence. Let us now consider the same test set as before, but we randomly inject a single exogenous event in each MAP. The goal is to assess how well the three strategies behave when partial observability and exogenous events combine together. Figure 17 shows the competence of the three strategies in such a faulty setting under the three observability levels. When the environment is fully observable, the three strategies behave exactly the same, as expected. Of course, the percentages of performed actions and achieved goals depend on how early, or how late, the exogenous event occurs in the MAP. In general, we can say that at least 70% of the actions are performed despite the injected exogenous event. A similar consideration can be made for the percentage of achieved goals.

When the observability conditions degrade to *HIGH* and *LOW*, however, it is easy to see that CWCM outperforms the other two strategies. This means that CWCM is actually more tolerant than the other strategies to partial observability even in the faulty scenario.

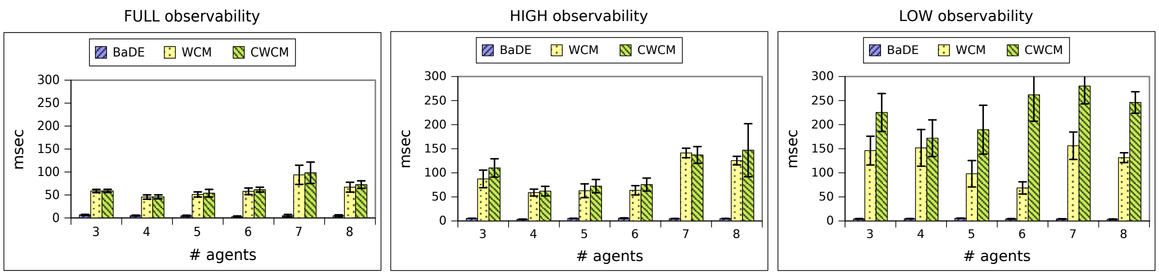

Figure 18: [*Faulty Conditions*] Monitoring time (average and 95% confidence interval) for a single execution step.

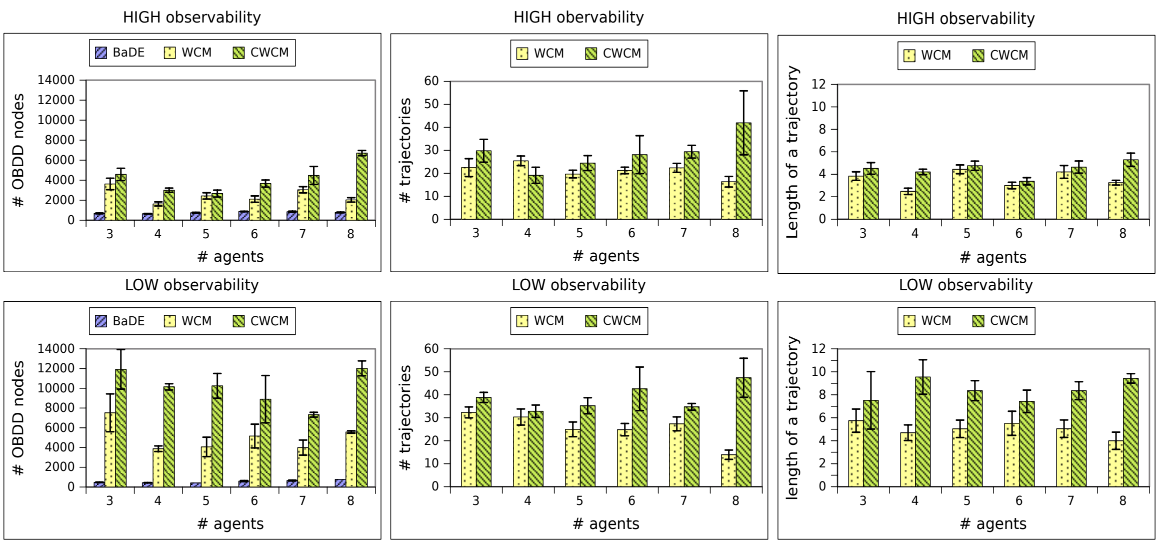

Figure 19: [*Faulty Conditions*] Average sizes of OBDDs encoding a trajectory-set (left), average number of trajectories within a trajectory-set (center), average length of a trajectory-set (right).

In particular, since the only difference between CWCM and WCM is the cooperative monitoring, we can conclude that the cooperation among the agents is actually beneficial.

Computational time. Figure 18 reports the computational cost, in milliseconds, of the three strategies under faulty conditions and for the three levels of system observability. It is important to note that also in this case the computational time strongly depends on observability level; whereas it does not depend on the number of agents in the team, nor on the presence of an exogenous event. In fact, the time for monitoring a MAP affected by an exogenous event has the same order of magnitude as the monitoring of a MAP under nominal conditions. The differences that can be observed by comparing charts in Figure 14 with charts in Figure 18 are due to the fact that the execution of a MAP affected by a fault terminates earlier than a MAP executed under nominal conditions, independently on the level of observability.

Of course, the BaDE strategy is the cheapest of the three, but it is unable to monitor effectively the execution of a MAP. In fact, the strong committed policy at the basis of this

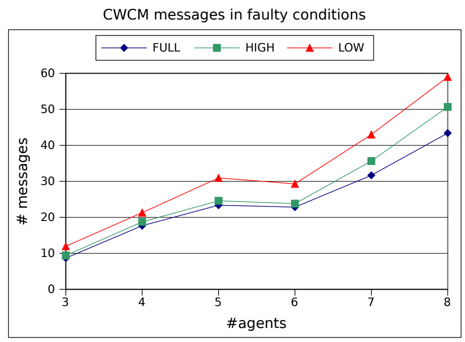

Figure 20: The number of messages exchanged by CWCM agents in faulty conditions.

strategy is too sensitive to the level of observability, and under *HIGH* and *LOW* conditions it performs poorly.

OBDD dimensions. Let us consider the dimensions of the OBDDs maintained by the three strategies. On the left-hand side of Figure 19, we report the sizes, in number of nodes, of the OBDDs representing the current belief state (BaDE strategy), and the current trajectory-set (WCM and CWCM strategies) under the three conditions of observability. As expected, BaDE keeps the smallest OBDDs since it just maintains the last belief state, but this makes the BaDE strategy unable to deal with low observability levels. WCM and CWCM behave similarly under *FULL* observability conditions, but CWCM tends to maintain bigger OBDDs when the observability level decreases. This result can be explained by the fact that CWCM can build longer trajectory-sets than WCM (Figure 19, right), and these longer trajectory-sets tend to be more ambiguous as demonstrated by the average number of trajectories within a trajectory-set (Figure 19, right).

CWCM Communication Analysis. Figure 20 shows the number of messages exchanged by CWCM agents under faulty conditions. The trend is similar to that of nominal conditions, however, the number of messages is slightly lower. This happens because the occurrence of a failure prevents the agents from performing some actions, and as a consequence some messages will not be exchanged. This is also the reason for having slightly less messages in SCN6 than SCN5. In fact, the number of inter-agent causal links in the two scenarios is almost the same, but faults in SCN6 have a stronger impact than in SCN5, this is evident looking at the number of performed actions in SCN5 and SCN6 (see Figure 17).

6.5 Experimental Analysis: Diagnosis

Competence. The competence of the diagnostic inferences is evaluated as the percentage of cases in which the action affected by the injected exogenous event has been included within the set of preferred explanations $mPADs$. Figure 21 (left-hand side) shows how our diagnostic inferences behave in the three levels of observability. Obviously, under *FULL* observability, the diagnostic inferences always identify the correct primary action failure. Under *HIGH* and *LOW* observability, however, the impaired agent can stop the plan execution due to lack of observations (i.e., not-enough-info). In those cases the diagnosis cannot identify the primary failure. Figure 21 (right-hand side) shows also the average distance (i.e., number of actions), between the action affected by an exogenous event, and the action

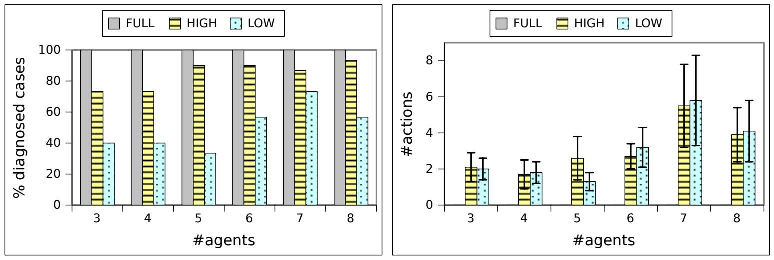

Figure 21: [*Diagnosis*] Competence (left), and responsiveness (right).

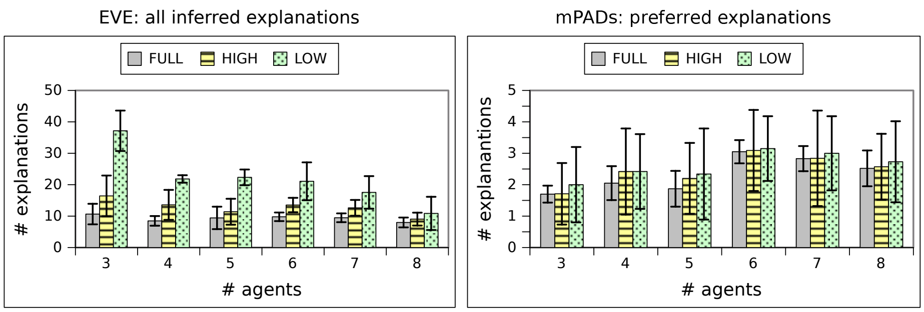

Figure 22: [*Diagnosis*] *EVE* explanations (left), *mPADs* explanations (right).

in which the failure is actually detected. Under *FULL* observability, the diagnosis is highly responsive as it detects an action failure as soon as the exogenous event occurs (i.e., the distance is zero). On the other hand, when the observability is just partial, a CWCM agent can take longer to detect a failure.

Explanations and Preferred Explanations. In Section 5, we have pointed out that, given a trajectory-set, one can identify two types of explanations: *EVE* and $mPADs$. The set *EVE* represents all the explanations that are consistent with the observations received by an agent; whereas $mPADs$ is the set of primary action failures inferred from EVE. Figure 22 shows the cardinalities (on average) of the two sets inferred in the six scenarios and with different levels of observability. From the two charts in Figure 22 we can draw two conclusions. First, the cardinality of *EVE* strongly depends on the observability level; namely, the reduction in the observability level causes an increment in the number of possible explanations. However, the cardinality of $mPADs$ is almost independent of the observability level. In fact, the number of preferred explanations inferred with *LOW* observability is similar to the number of preferred explanations inferred with *FULL* observability in all the six scenarios. Of course, $mPADs$ sets computed under *LOW* observability tend to be slightly bigger than $mPADs$ sets computed under *FULL* observability, but this is a consequence of the fact that the initial *EVE* set was more ambiguous under *LOW* observability. This means that, regardless of the initial ambiguity of the *EVE* sets, the preferred explanations are reduced to almost the same subsets in all the six scenarios.

The second important conclusion is that $mPADs$ explanations are substantially more useful in identifying a fault than *EVE* explanations. In fact, the average cardinality of $mPADs$ sets is three in the worst cases with *LOW* observability; whereas, the average

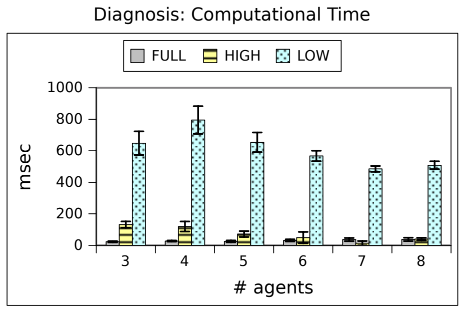

Figure 23: Diagnosis: Computational Time.

number of EVE explanations in the best case with $FULL$ observability is eight, but it rises up to 38 in the worst case with LOW observability (see SCN3). This means that the $mPADs$ explanations may actually help a human user refine her/his hypotheses about the current situation of the system. This is essential when we consider that diagnosis is just the first step for recovery (Micalizio, 2013). Thus, a human user, or possibly an automatic supervisor, has to consider a small number of alternative explanations, and hence can better focus the plan recovery process on the fault(s) that are believed more plausible.

Computational Effort. Finally, we consider the computational time required to infer the diagnoses. In inferring the EVE explanations, the computational cost is mainly due to the cost of removing non-relevant variables from the trajectory-set provided by CWCM (see the Appendix for a discussion from a theoretical point of view about the cost of such variable removal). Figure 23 reports the average computational time, in milliseconds, for extracting EVE explanations in the six scenarios and with the three different levels of observability. As noticed in the previous section, under LOW observability, the trajectory-set tends to be bigger than with the other two observability levels. As a consequence the time for inferring diagnoses under LOW conditions tends to be higher; however, this time is below 1 second even in the worst case. On the other side, under $HIGH$ observability conditions, the worst time is below 150 milliseconds (see scenario SCN3). The worst time falls below 50 milliseconds when we consider the $FULL$ observability level. These computational times allow us to conclude that the diagnostic task, as the monitoring one, can be performed on-line in a number of applicative domains where actions are performed in the order of seconds, or even minutes.

6.6 Discussion

At the beginning of this experimental analysis we posed three questions, and now we are in the position to answering them. First of all, the experimental results show that CWCM is not sensitive to number of agents in the team. This is a consequence of the partitioning of the global plan into local plans. In such a way, in fact, each agent keeps just its own point of view on the states of the shared resources; namely, each agent has a local belief state that does not depend on the number of agents in the team. As we have seen, the consistency among these local beliefs is guaranteed through the exchange of messages whose number is linear in the number of inter-agent causal links of the MAP under consideration.

On the other hand, the level of observability of the system has a strong impact both on the computational effort of CWCM and on the ambiguity of the trajectory-sets computed. The lower the observability, the higher is the computational cost and the bigger is the trajectory-sets. An important result that emerges from our analysis is that the worst-case scenarios depicted in the Appendix are very rare, and never occurred during our experiments. Indeed, the compact encoding of trajectory-sets and action models obtained via OBDDs facilitates a very efficient implementation of CWCM that takes, on average, just hundreds of milliseconds to monitor a single action. This allows us to conclude that CWCM can be successfully employed for on-line monitoring in many real-world domains.

The level of observability has also an impact on the diagnostic inferences. In fact, the number of EVE explanations significantly grows as the observability level decreases. However, the number of preferred $mPADs$ explanations is not so strongly influenced by the observability level.

Finally, the direct comparison between CWCM and WCM demonstrates that the cooperation among the agents is essential to be tolerant to very scarce observations. The cooperation, in fact, is the means through which an agent in CWCM can keep longer trajectory-sets than in WCM. These longer trajectory-sets give each agent more chances to collect pieces of information about the successful completion of some pending actions.

7. Related Works

We consider four main families of model-based approaches to the diagnosis of dynamic systems close to our MAPs:

- Discrete-Event Systems (DESs);
- Relation-oriented;
- Team-oriented;
- Resource-oriented.

In the rest of this section we briefly review the main approaches within these families, highlighting differences and similarities with the CWCM methodology proposed here.

7.1 Discrete-Event Systems

Since the seminal work by Sampath, Sengupta, Lafortune, Sinnamohideen, and Teneketzis (1995) on the Diagnoser, a huge number of works have addressed the diagnosis of dynamic systems by modeling such systems as DESs. While the Diagnoser approach compiles the diagnostic model (i.e., the Diagnoser itself) of the whole system off-line, other approaches (see e.g., Lamperti & Zanella, 2002; Cordier & Grastien, 2007) compute all possible system behaviors, and check which of these behaviors are correct. Grastien, Haslum, and Thiébaux (2012) extends to DESs the conflict-based approach initially proposed by Reiter (1987) on static systems.

To the best of our knowledge, the DES framework that gets closer to ours is the one presented by Grastien, Anbulagan, Rintanen, and Kelareva (2007). In such a framework, a diagnosis is a label either *normal* or *faulty*, associated with a system trajectory; where a trajectory is a sequence of system states interleaved with events, thus very similar to the trajectories kept within our trajectory-sets. A trajectory is *normal* if it does not contain fault events; the trajectory is *faulty*, otherwise.

Grastien et al. propose to reduce a diagnosis problem to a SAT one. The idea is to formulate a SAT problem, in order to answer the question "is the observed behavior compatible with at most i faults occurring?". Of course, when the answer is yes for $i = 0$, the system is assumed nominal as there exists at least one normal path consistent with the observations. In principle, the proposed system description could encode a MAP: the execution of actions could be modeled by a subset of observable events; whereas our exogenous events could be mapped to unobservable events directly. However, the DES framework cannot be directly applied to the same domains CWCM can deal with. First of all, in the DES approach the next state of the whole system is inferred taking into account the synchronous occurrence of a set of events. Thus, if agents are event generators, it follows that they can only perform actions synchronously, but in CWCM this restriction is not imposed. Moreover, the SAT-based methodology is centralized as trajectories are about whole system states, whereas CWCM enables each agent to build local trajectory-sets in a distributed way.

7.2 Relation-Oriented Approaches

Relation-oriented approaches have been proposed by Micalizio and Torasso (2008, 2009). We define these works as relation-oriented since action models are expressed in terms of relations over agents' state variables. The advantage of such a kind of model is the possibility of representing in a single piece of knowledge both the nominal and the abnormal evolutions of an action. The CWCM methodology falls within such a category, and extends the previous works in two ways. First of all, CWCM is able to deal with completely unexpected events, denoted as $\xi_\star$, for which no model exists. Indeed, the occurrence of $\xi_\star$ during the execution of an action a maps all variables in **effects**(a) to the *unknown* value; meaning that these variables are no longer predictable.

The second important extension is the protocol that allows the agents to cooperate with each other during the monitoring task. As the experimental results have demonstrated, cooperation among agents is essential to cope with very scarce observations. By means of the cooperation, in fact, an agent can acquire new pieces of information that it would not acquire directly. These further pieces can therefore be used to refine its own trajectory-set, and possibly the outcome of some pending actions could be determined.

7.3 Teamwork-Oriented Approaches

Rather than diagnosing action failures, as in CWCM, teamwork-oriented approaches are focused on diagnosing teamwork failures; i.e., *coordination failures*. This type of failures are not necessarily due to erroneous actions, but to wrong decisions taken by the agents.

The detection of teamwork failures has been addressed in some seminal works by Tambe (1998) and Kaminka and Tambe (2000). Kalech and Kaminka (2003) have later focused on the diagnosis of these coordination failures, and have introduced the notion of *social diagnosis*. More specifically, the team of cooperating agents is represented in abstract terms by means of a hierarchy of *behaviors*. A behavior is an abstraction of the concrete actions that an agent actually takes in the real world. Indeed, behaviors abstract not just single actions, but possibly sequences of actions. Thus, differently from relational- and resource-oriented approaches (see later), an explicit model of the agents' plans is missing in teamwork-oriented solutions.

The social diagnosis framework assumes that agents synchronize themselves to jointly select a team behavior. A disagreement arises when at least two agents select two behaviors that are incompatible with each other. Such disagreements represent instances of social diagnosis problems. Of course, agents select their behaviors according to their own beliefs, thus a social diagnosis for a disagreement is a set of conflicting belief states held by a subset of agents. Kalech and Kaminka (2005, 2007, 2011) propose different methods for inferring a social diagnosis. These solutions, however, rely on some assumptions that may limit their applicability in real-world scenarios. First of all, it is assumed that all the agents in the team share the hierarchy of behaviors and that the belief states of the agents are homogeneous (i.e., defined over the same set of propositional atoms). Moreover, agents must be willing to exchange each other their own beliefs. The CWCM methodology we propose, however, does not suffer from these limitations. CWCM, in fact, makes no assumption about the agents' internal beliefs. In addition, the communication among the agents does not exchange agents' internal beliefs, but observations about shared resources that agents directly gather, and this guarantees the agents a high degree of privacy.

7.4 Resource-Oriented Approaches

The approaches within the resource-oriented family have mainly been proposed by Roos and Witteveen. We call their approaches resource-oriented because, from their point of view, the system to be diagnosed is a plan, and the state of such a system is given by the states of the system resources. The execution of an action can just change the state(s) of one or more resource(s). These approaches deserve particular attention as they have some similarities with the CWCM methodology, but there are also a number of relevant differences.

Witteveen, Roos, van der Krogt, and de Weerdt (2005) present the basic framework that they use and extend in their subsequent works. In this framework, actions are modeled as atomic plan steps; more precisely, action models are functions that deterministically map resource states in input into resource states in output. These models therefore represent just the changes *normally* caused by actions when they are successfully performed. The faulty behavior of actions, conversely, is modeled via a very weak *abnormal function*, which maps the state of each resource in input to the *unknown* value. This means that, when an action fails, the states of the resources handled by that action become unpredictable.

A diagnostic problem arises when the observations received at an execution step are inconsistent with the nominal predictions made through the action models. This means that at least one of the actions performed so far behaved abnormally. Witteveen et al. (2005) introduce the notion of *plan diagnosis* as a subset of plan actions that, once qualified as abnormal, make the observations consistent with the predictions made assuming all the other actions as nominal.

Of course, since there may exist many possible plan diagnoses, it is important to look for diagnoses that are more preferable than others. Roos and Witteveen (2009) propose a different preference criterion based on the predictive power that each plan diagnosis has. They therefore introduce the notion of *maximally-informative plan diagnosis* (maxi-diagnosis) as a set of plan diagnoses that predict correctly the biggest subset of observations. This notion of diagnosis is subsequently refined by the notion of *minimal maximally-informative plan diagnosis* (mini-maxi-diagnosis), which is just a subset of maxi-diagnosis such that the number of failed actions to be assumed is minimal.

In the work by de Jonge et al. (2009), the basic framework is extended: agents are seen as resources, and action models also includes variables about agents' equipment and environment events (i.e., exogenous events). This extension allows the distinction between *primary* and *secondary* diagnoses. While a primary diagnosis is a plan diagnosis (i.e., expressed in terms of failed actions), a secondary diagnosis can be thought of as a second level diagnosis that tries to explain why a given action failure has occurred.

CWCM tries to resolve the same problem as the one addressed by Roos and Witteveen (2009): Diagnosing the execution of a MAP. However, action models are significantly different in the two approaches. From Roos and Witteveen's point of view, action models are deterministic functions of nominal behavior only. Whereas in CWCM, we model actions as relations that easily accommodate both nominal and faulty evolutions. In particular, faulty evolutions can be nondeterministic, and just partially specified as they support the *unknown* value to indicate that no expectations are possible for a given variable.

Another important difference between the two approaches is about the execution of actions. Roos and Witteveen assume that actions take just one time instant to be performed and that action execution proceeds synchronously all over the agents. Our CWCM is more realistic since action execution is asynchronous: even though actions are modeled just in terms of preconditions and effects, their actual duration is not necessarily one time instant. As we have seen, in fact, agents cooperate with each other by exploiting the causal and precedence links that are explicitly defined within our plan model. The plan model adopted by Roos and Witteveen, instead, mentions explicitly precedence links only, while it does not include causal links.

Also the process with which a diagnosis is inferred presents substantial differences. Witteveen et al. (2005) and de Jonge et al. (2009) present a centralized method to carry out diagnostic inferences. A distributed procedure for qualifying actions as abnormal is proposed by Roos and Witteveen (2009), but also in this case the detection of a diagnostic problem is made in a centralized way. Moreover, the methodology proposed by Roos and Witteveen is a sort of strong committed approach, in the sense that whenever there are some observations, the system has to infer a diagnosis. On the other hand, our CWCM methodology is fully distributed both in the detection of a diagnostic problem (i.e., monitoring), and in its solution. In addition, CWCM is inherently weak committed: observations do not necessarily trigger a diagnostic process, but diagnosis inferences start after an interpretation of the observations that either lead to (1) determining an action failure, or (2) determining that a service produced in favor of another agent's action is actually missing. CWCM achieves this second point by exploiting both direct observations gathered by the agent, and messages coming from other agents. This means that when observations are not sufficient to either reach condition (1) or (2), a diagnosis is not inferred.

As said above, de Jonge et al. (2009) introduce a distinction between primary and secondary diagnosis. Such a distinction can also be found in our methodology. The primary diagnosis by de Jonge et al. corresponds to our minimum primary action failures ($mPADs$), which identify the actions that should be assumed faulty in order to make the plan execution consistent with the observations. The secondary diagnosis, on the other hand, corresponds to our refined explanations (*refinedExp*), in which we associate each action in $mPADs$ with a set of exogenous events that, consistently with the observations, might have occurred and hence caused the action failure.

In this paper we also assess the impact of a primary action failure $a \in mPADs$ by inferring the set of secondary action failures; namely, the subset of actions that fail as an indirect consequence of the failure of a. Although the identification of secondary failures would be possible, de Jonge et al. do not take into account the problem.

In conclusion, the CWCM framework can be considered as an extension of the frameworks by de Jonge et al. (2009) and Roos and Witteveen (2009). In fact, the action models proposed by Roos and Witteveen can be reproduced within our framework by including in each relation-based model just two entries: one for the deterministic nominal evolution of the action, and one for the abnormal behavior where all the agent variables become *unknown* as a consequence of an unpredictable event. These action models could be used by CWCM as usual to infer a plan diagnosis in a fully distributed way.

8. Conclusion

Plan diagnosis is an essential step to implement robust multiagent planning systems. As shown in other works (Mi & Scacchi, 1993; Gupta et al., 2012; Micalizio, 2013), in fact, the explanations provided by a plan diagnosis can steer a repair procedure and make the repair process more effective.

In this paper we have addressed the problem of plan diagnosis by splitting it into two subproblems: the detection of action failures, and the actual explanation of the detected action failures in terms of exogenous events that might have occurred. The detection of action failures is achieved by means of the *Cooperative Weak-Committed Monitoring* (CWCM) strategy, which allows agents to cooperate with each other during the monitoring task. Cooperation among agents plays a central role not only for the detection of action failures, but also for their explanations. The CWCM methodology, in fact, allows each agent to build a structure (i.e., a trajectory-set), that is an internal representation of the world from the point of view of the agent itself. Relying on this structure, each agent can infer explanations for its own action failures without the need of interacting with other agents.

The proposed framework to the diagnosis of MAPs extends previous approaches in literature. First of all, CWCM is fully distributed and asynchronous. Previous approaches (see e.g., Kalech & Kaminka, 2011; Roos & Witteveen, 2009; Micalizio & Torasso, 2008), instead, are all based on some synchronous step (e.g., agents execute actions synchronously). In our framework an agent can perform its next action as soon as the action's preconditions are satisfied. To verify this condition, we just impose that agents adhere to a coordination protocol that guarantees the consistent access to the shared resources.

In addition, we propose here an extension to the relational language for modeling nondeterministic actions (Micalizio & Torasso, 2008). In the previous approach, in fact, we assume to know in advance all the exogenous events that can affect a given action; in this paper we are able to deal with partial knowledge about the exogenous events. In particular, we allow to specify just a subset of the effects an exogenous event has on an action (i.e., some agent's variables might become *unknown* after the event), but we also allow to specify that an action might be affected by an indefinite event whose effects are completely unpredictable (i.e., all agent's variables become *unknown* due to the event). This kind of extended action model subsumes the action models proposed by Roos and Witteveen, which just consists of two parts: the nominal action model, and an abnormal model that maps each agent's variable into the *unknown* value.

Cooperation among agents and nondeterministic action models make CWCM particularly apt to deal with dynamic and partially observable environments. On the one side, the nondeterministic action models we have discussed here capture unexpected changes in the environment. On the other side, the cooperative monitoring allows each agent to acquire information about the environment from the other agents. It is important to note that, differently from other works where agents exchange each other their internal belief states (see e.g., Kalech & Kaminka, 2011), in CWCM an agent just needs to communicate what it observes. This enables agents to keep private their internal beliefs; in addition, agents could adopt specific policies for deciding what observations should be forwarded to what agents. Forwarding some observations to more agents, and not just to a single agent as in the current proposal, might help the agents to discover earlier the outcomes of some pending actions; we leave this opportunity for future research.

It must also be noted that CWCM just assumes that observations are *correct*: The actual state of an agent must not be pruned off the agent belief state due to an erroneous observation. This assumption is often made also in many model-based approaches to diagnosis (see e.g., Birnbaum et al., 1990; Brusoni, Console, Terenziani, & Theseider Dupré, 1998; Pencolé & Cordier, 2005; Roos & Witteveen, 2009; Eiter, Erdem, Faber, & Senko, 2007, just to mention a few). Correctness of the observations, however, does not implies that observations must be precise. CWCM can in fact consume ambiguous messages given as a disjunction of values for the same variable (i.e., $var = v_1 \vee var = v_2 \vee \ldots var = v_n$), or as a negation of a specific value (i.e., $var \neq v$). From the point of view of CWCM, consuming such observations simply corresponds to the selection of states within the belief state the observations refer to. Although this aspect has not been emphasized in the paper, the ability of dealing with ambiguous observations enriches the communicative capacities of the agents. For instance, in an *ask-if* interaction, a client, rather than answering with a generic *no-info*, could give the provider a disjunction of possible resource states among which, however, the client is incapable to discriminate the actual one. This set of alternative states is, from the point of view of the provider, much more informative than *no-info*, and possibly could lead the provider to determine the actual state of the resource at hand.

From the point of view of the diagnostic inference, we have shown that it is possible to explain action failures by extracting explanations from the trajectory-sets built by CWCM. In particular, we have pointed out that assuming action failures independent of each other might lead to spurious diagnoses. For this reason we have proposed a methodology for identifying *primary action failures* and *secondary action failures*, which are just an indirect consequence of the primary ones. A simple preference criterion, based on the minimality of the primary action failures, has been proposed to prefer alternative explanations.

A deep experimental analysis has shown that both the cooperative monitoring and diagnosis are practically feasible. An efficient implementation based on OBDDs is discussed in the Appendix together with a computational analysis from a theoretical point of view. The experiments have highlighted that CWCM scales up well with the number of agents, but it is affected by the level of observability of the environment: the trajectory-sets tend to be bigger as the environment is less observable. However, the experiments demonstrate that the cooperation is effective even in dealing with very scarcely observable environments. Competence rates for noncooperative solutions, in fact, are comparable with those of CWCM only when the environment is fully observable; in other situations, instead, CWCM always exhibits the highest competence.

The proposed framework can be extended in different ways. As mentioned above, we have so far adopted a careful approach to communication by restricting the agents to talk with each other only about the exchanged services. However, the agents might be willing to communicate further pieces of knowledge they have acquired. An interesting possible extension is to improve the cooperative protocol along this direction. The intuition, in fact, is that when an agent acquires more information, it could infer the outcome of some of its pending actions earlier than what it does now. Of course, the communication must not become a bottleneck, so agents should be able to identify what piece of information is worth to be forwarded to what agents, and avoid broadcasting every observation to all the agents.

The most important extension we aim at, however, is to relax the assumption that communication among the agents is always reliable. Removing such an assumption has many consequences. First of all, the cooperative monitoring protocol should be extended in order to deal with messages that can be lost. Moreover, Proposition 7, about the safe use of resources, might no longer be guaranteed by CWCM; thus resources could be accessed inconsistently. To diagnose these situations we could take a point of view similar to Kalech and Kaminka's social diagnosis. In fact, erroneous access to resources, could be considered as coordination failures. This would impact the diagnostic inferences that should no longer be local, but distributed. That is, as for the monitoring task, also diagnosis should be performed by means of the cooperation of a number of agents.

Acknowledgments

The authors wish to thank the anonymous reviewers for their insightful comments, that have substantially contributed to the final shape of this work.

Appendix A. Implementation and Computational Analysis

In this Appendix we first recall some basic OBDD operators and their complexities, and then we study the computational cost of the most expensive relational operations involved by the CWCM and diagnostic methodologies discussed above.

A.1 OBDD Operators and their Complexities

The computational analysis we discuss in the next subsection relies on the results presented by Bryant (1986, 1992). In these works, the author discusses an efficient implementation of OBDDs operators and their corresponding computational complexities. These results are summarized in Table 6, where f, f_1, and f_2 denote the Boolean functions, encoded by the reduced function graphs G, G_1, and G_2, respectively. The size of graph G corresponds to the number of its vertices, and it is represented as $|G|$. The primitive OBDD operators are reported in the upper side of Table 6:

- *reduce* builds the canonical form of a Boolean function f; i.e., given a specific variables ordering, the reduce operator gets a graph G whose size is minimal.
- *apply* implements binary logical operations between two Boolean functions f_1 and f_2; the operator works on graphs G_1 and G_2 encoding the two functions, respectively; `op` can be any binary logical operator $(\wedge, \vee, \rightarrow, \leftrightarrow)$. The computational complexity in the worst case is O the product of the sizes (i.e., number of vertices) of the two graphs.
- *restrict* substitutes a constant b to a variable x_i in time almost linear in the number of vertices within the graph G.
- *rename* renames a set of variables $\vec{x}$ with a new one $\vec{x}'$; its complexity is exponential in the number of renamed variables.
- *equiv* checks the equivalence of the two Boolean functions f_1 and f_2; since this operator scans the two corresponding graphs simultaneously, the computational complexity is linear in their sizes.

In the lower side of Table 6 we report the computational cost in time and space for the relational operators JOIN, INTERSECT, UNION and PROJECT that can be obtained by combining primitive OBDD operators. Observe that, among the relational operators, the projection is the most expensive; in fact, it is exponential in the number of the (binary) variables to be removed (see e.g., Torta & Torasso, 2007; Torasso & Torta, 2006 for details).

A.2 CWCM: Computational Analysis

To analyze the computational complexity of CWCM, we consider the high-level algorithm presented in Figure 10, and focus on the computational cost of performing a single iteration of the while loop when an action a_l^i is actually performed in the real-world. In such a situation there are three main steps that hide potentially expensive operations on relations:

- the extension of the current trajectory-set (line 9);
- the refinement of the trajectory-set with the available observations (line 11);
- the detection of the outcomes of the pending actions (line 19, Figure 7).

In the rest of this section we analyze the computational effort of each of these steps.

operator	**time**	**size**
reduce(f)	$O(\lvert G\rvert \cdot \log \lvert G\rvert)$	$\lvert G\rvert$
apply(op, f_1, f_2)	$O(\lvert G_1\rvert \cdot \lvert G_2\rvert)$	$\leq \lvert G_1\rvert \cdot \lvert G_2\rvert$
restrict(x_i, b, f)	$O(\lvert G\rvert)$	$\leq \lvert G\rvert$
rename (f, $\vec{x}$, $\vec{x}'$)	$O(\lvert G\rvert \cdot 2^{\lvert\vec{x}\rvert})$	$\leq \lvert G\rvert \cdot 2^{\lvert\vec{x}\rvert}$
equiv(f_1, f_2)	$O(max(\lvert G_1\rvert, \lvert G_2\rvert))$	N/A
join (f_1, f_2); union(f_1, f_2);	$O(\lvert G_1\rvert \cdot \lvert G_2\rvert)$	$\leq \lvert G_1\rvert \cdot \lvert G_2\rvert$
intersect(f_1, f_2) (i.e., select)	$O(\lvert G_1\rvert \cdot \lvert G_2\rvert)$	$\leq \lvert G_1\rvert \cdot \lvert G_2\rvert$
project(f, $\{x_1, \ldots, x_n\}$, $\{y_1, \ldots, y_m\}$)	$O((2^{(n-m)} \cdot \lvert G\rvert)^2)$	$\leq 2^{(n-m)}\lvert G\rvert$

Table 6: OBDD operators and their complexity.

A.2.1 Extending the Trajectory-Set

According to equation (7), the operator $\otimes$, which from $Tr^i[1,l]$ yields a new trajectory-set $Tr^i[i,l+1]$, involves two join operations: one between $Tr^i[1,l]$ and $\Delta(a^i_l)$, and one between $Tr^i[1,l]$ and $\Gamma(a^i_l)$. The results of these two operations are subsequently merged into a new trajectory-set $Tr^i[1,l+1]$ via a union operation. To understand the computational cost of these relational operations, it is necessary to map them into OBDD operators. As already shown in previous works (see e.g., Torta & Torasso, 2007; Micalizio, 2013), the natural join can be mapped to the *and* between two Boolean functions (and hence between two OBDDs), whereas the union of two relations becomes the Boolean *or*. Let G_l, G_{l+1}, G_Δ, and G_Γ be the OBDDs corresponding to the relations $Tr^i[1,l]$, $Tr^i[1,l+1]$, $\Delta(a^i_l)$, and $\Gamma(a^i_l)$, respectively; the $\otimes$ operator can be mapped to the following expression in terms of OBDDs operations:

$$G_{l+1} = apply(\vee, apply(\wedge, G_l, G_\Delta), apply(\wedge, G_l, G_\Gamma)) \tag{14}$$

Given the operator complexities in Table 6, the computational effort to infer the new trajectory-set is in the worst case:

$$\begin{aligned} O(|G_{l+1}|) &= O(|G_l| \cdot |G_\Delta|) \cdot O(|G_l| \cdot |G_\Gamma|) \\ &= O(|G_l|^2 \cdot |G_\Delta| \cdot |G_\Gamma|). \end{aligned} \tag{15}$$

A.2.2 Refinement with Observations

Once the new trajectory-set has been inferred, it is refined with the observations obs^i_k received by the agent. For the sake of exposition, in equations (8) and (9) we defined the refinement of a trajectory-set as an intersection between the trajectory-set itself and the belief state $\mathcal{B}^i_k$ refined with obs^i_k. Note that the extraction of a belief state is a very expensive operation, thus we try to avoid this operation whenever possible. In this particular case, since the agent variables VAR^i_k are included within the current trajectory-set, the refinement operation can be carried out as an intersection between $Tr^i[1,l+1]$ and obs^i_k; in terms of OBDD operators:

$$G_{ref.l+1} = apply(\wedge, G_{l+1}, G_{obs^i_k}) \tag{16}$$

where $G_{obs^i_k}$ is the OBDD encoding obs^i_k, and $G_{ref.l+1}$ is the OBDD corresponding to the refined trajectory-set *refinedTr*$^i[1,l+1]$. It follows that the computational cost of this operation is

$$O(|G_{ref.l+1}|) = O(|G_{l+1}| \cdot |G_{obs^i_k}|) \tag{17}$$

A.2.3 Detecting Pending Actions' Outcomes

The last step of the CWCM algorithm we consider is the assessment of the outcome of every action currently within the list of pending actions $pActs^i$. In Section 4, we noted that to verify the success of a given action $a^i_k \in pActs^i$, it is sufficient to check whether the nominal effects of a^i_k are satisfied in every state in $\mathcal{B}^i_{k+1}$ (Definition 3). In case such a condition does not hold, one has to verify whether the expected nominal effects of a^i_k are missing in each state of $\mathcal{B}^i_{k+1}$ (Definition 4). If both checks result in a negative answer (i.e., the belief state $\mathcal{B}^i_{k+1}$ is still too ambiguous), action a^i_k remains pending.

Extracting the belief state $\mathcal{B}^i_{k+1}$ for explicitly checking these conditions might be particularly expensive, especially when the trajectory-set grows over time. The extraction of a belief state through the PROJECT operation, in fact, would require the elimination from $Tr^i[1, l+1]$ of all the variables we are not interested in; thus, we would remove the variables about steps $1, 2, \ldots, k, k+2, \ldots, l+1$, that is $l \cdot |VAR^i|$ variables. As Table 6 shows, the complexity of PROJECT is exponential in the number of variables to be removed, so it could easily become a bottleneck.

To cope with this problem, we implemented the checking for the *ok* and $failed$ outcomes in a different way. In particular, from Definition 3 it directly follows:

Proposition 11 *a^i_k has outcome ok iff $\mathcal{B}^i_{k+1}$* JOIN **effects**(a^i_k) *equals B^i_{k+1}.*

Proof: The proof is straightforward: by definition a^i_k has outcome *ok* iff its nominal effects are satisfied in every state in $\mathcal{B}^i_{k+1}$; the join between $\mathcal{B}^i_{k+1}$ and **effects**(a^i_k) yields $\mathcal{B}^i_{k+1}$ only when the nominal effects are already included in every state in $\mathcal{B}^i_{k+1}$, and hence the action has outcome *ok*. □

Since $\mathcal{B}_{k+1}$ is included in *refined*$Tr^i[1, +l+1]$, the above proposition can be extended to the whole trajectory-set:

Proposition 12 *a^i_k has outcome ok iff*

$$\textit{refined}Tr^i[1, l+1] \text{ JOIN } \mathbf{effects}(a^i_k)\, \textit{equalsrefined}Tr^i[1, l+1].$$

That is to say, after the refinement of the trajectory-set with the observations, action a^i_k has outcome *ok* iff its nominal effects do not filter out any trajectory from *refined*$Tr^i[1, l+1]$. Relying on this proposition, we can verify whether a^i_k has outcome *ok* in two steps: first, we build a temporary OBDD maintaining the result of the join between $Tr^i[1, l+1]$ and **effects**(a^i_k), and then we check whether the temporary OBDD is equivalent to the original trajectory-set; in terms of OBDD operators:

$$\text{outcomeOK?} = equiv(G_{ref.l+1}, apply(\wedge, G_{ref.l+1}, G_{\mathbf{effects}(a^i_k)})) \tag{18}$$

Since the size of $G_{\mathbf{effects}(a^i_k)}$ is negligible when compared to the size of $G_{ref.l+1}$, computational complexity of this check is about $O(|G_{ref.l+1}|^2)$.

The $failed$ outcome can be checked in a similar way; in this case we want to discover whether the nominal effects of a^i_k are missing in $Tr^i[1, l+1]$; this happens only when the negation of the effects $\overline{\mathbf{effects}(a^i_k)}$ do not represent a possible filter for $Tr^i[1, l+1]$ so that $Tr^i[1, l+1]$ JOIN $\overline{\mathbf{effects}(a^i_k)}$ is again equals to $Tr^i[1, l+1]$. In terms of OBDD operators

$$\text{outcomeFailed?} = equiv(G_{ref.l+1}, apply(\wedge, G_{ref.l+1}, G_{\overline{\mathbf{effects}(a^i_k)}})) \tag{19}$$

It is important to note that the OBDD $G_{\overline{\mathbf{effects}(a^i_k)}}$ is computed in constant time directly from $G_{\mathbf{effects}(a^i_k)}$; in fact, given a Boolean function f and the corresponding graph G_f, it is sufficient to exchange with each other the **0** and **1** nodes in G_f to obtain the graph representation of the Boolean function $not(f)$. Thus, also this check is about $O(|G_{ref.l+1}|^2)$.

It follows that the cost for determining the outcomes of the actions in $pActs^i$ is:

$$O(|pActs^i| \cdot |G_{(ref.)l+1}|^2). \tag{20}$$

From equation (20), it is easy to see that the computational cost of the CWCM methodology strongly depends on the amount of available observations. The worst case is in fact when at a give step l, due to very scarce observations, the number of pending actions is close to l itself; that is, $|pActs^i| \approx l$, meaning that almost all the actions performed so far have outcome *pending*.

A.3 Diagnosis: Computational Analysis

The computational cost of the diagnostic process is strongly dominated by the cost for inferring the event-based explanations (EVE). As we have shown, in fact, it is possible to extract the set of minimum cardinality primary action failures explanations ($mPADs$) from such a structure. According to Equation 11, the EVE set can be extracted as a projection of the current trajectory-set $Tr^i[1, l]$ over the event variables $e_1, \ldots, e_{l-1}$; unfortunately, in this case there is no way to avoid this expensive operation.

To estimate its computational cost, we have first to consider how many *binary* variables are within the OBDD G_l encoding $Tr^i[1, l]$, and how many (binary) variables we are going to remove from that OBDD. Each state and event variable in $Tr^i[i, l]$ is in fact a multi-valued variable that is actually implemented in terms of a number of binary variables within the OBDD G_l. The number of required binary variables depends on the size of the domain of the original high-level variable. Let us assume that d is the size of the largest domain of the variables in VAR^i, then we can estimate that we need $b = \log d$ binary variables for each variable mentioned in $Tr^i[1, l]$ (both state and event variables). It is easy to see that the number of binary variables required to represent a single belief state is $w = b * |VAR^i|$: for each multi-valued variable in VAR^i we have b binary variables at OBDD level.

The number of binary variables encoding the trajectory-set $Tr^i[i, l]$ is therefore $p = l \cdot w + (l - 1) \cdot b$; in fact, within $Tr^i[1, l]$, we have l beliefs and l-1 event variables. The cost of projecting $Tr^i[1, l]$ over the event variables is therefore:

$$O((2^{p-l \cdot w} \cdot |G_l|)^2). \tag{21}$$

Once EVE diagnoses have been extracted, it is possible to infer the minimum cardinality primary failures by exploiting techniques by Torasso and Torta (2003), which are proven to be polynomial in the size of the OBDD.

A.4 Discussion

A first important result that emerges from the computational analysis above is that the monitoring of a single execution step with CWCM is not exponential. In fact, we have shown that each step of its declarative definition can be mapped into a number of OBDD operators whose complexity is polynomial, provided that the sizes of the involved OBDDs remain manageable. In particular, we have shown that the only exponential operation used in the declarative definition, the projection, can be avoided in the actual implementation. The main concern with CWCM is that the trajectory-set may grow over time as an agent performs actions without receiving observations. Consequently, also the computational cost of CWCM tends to grow over time since the size of the OBDD encoding the trajectory-set may increase. It is important to note, however, that such a growth is not exponential but quadratic (see equation (15)). In addition, to estimate the computational costs of both monitoring and diagnosis, we exploited the estimations reported in Table 6; these, however, are estimates of the very worst possible cases, but in practice these cases are not very common. Bryant conjectures that, although the theoretical cost of the apply operator between two OBDDs G_1 and G_2 is $O(|G_1| \cdot |G_2|)$ in the worst case, in practice the actual cost is in most of the cases closer to $O(|G_1| + |G_2| + |G_3|)$ where G_3 is the resulting OBDD (Bryant, 1986). Thus also the size of the resulting, intermediate OBDDs plays a central role in determining the actual computational cost.

In the specific case of CWCM, we have to observe that it is not very common for an agent to perform a long portion of its plan without receiving observations. The CWCM allows in fact the agents to communicate with each other; therefore, unless an agent is completely isolated from others, each agent will likely receive observations coming from the other agents about the services it has provided them with. This means that, in practice, the size of the OBDD encoding the trajectory-set should not become intractable because of the cooperation among the agents, and the experiments we have conducted so far support this hypothesis.

On the other hand, the diagnostic inferences are slightly more expensive than the monitoring strategy. This because the project operation cannot be avoided in order to infer a diagnosis. In this case, however, we have to observe that the plan execution has already been stopped as a consequence of a detected failure. Thus, the diagnosis can take more time to infer a result since it is not constrained to be on-line.

References

Arasu, A., Babu, S., & Widom, J. (2006). The CQL continuous query language: semantic foundations and query execution. *The International Journal on Very Large Data Bases*, *15*(2), 121–142.

Birnbaum, L., Collins, G., Freed, M., & Krulwich, B. (1990). Model-based diagnosis of planning failures. In *Proc. of Association for the Advancement of Artificial Intelligence (AAAI'90)*, pp. 318–323.

Boutilier, C., & Brafman, R. I. (2001). Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, *14*, 105–136.

Brusoni, V., Console, L., Terenziani, P., & Theseider Dupré, D. (1998). A spectrum of definitions for temporal model based diagnosis. *Artificial Intelligence*, *102*, 39–79.

Bryant, R. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Bryant, R. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computer Surveys*, *24*, 293–318.

Cordier, M.-O., & Grastien, A. (2007). Exploiting independence in a decentralised and incremental approach of diagnosis. In *Proc. of the International Joint Conference on Artifical Intelligence (IJCAI'07)*, pp. 292–297.

Cox, J. S., Durfee, E. H., & Bartold, T. (2005). A distributed framework for solving the multiagent plan coordination problem. In *Proc. of International Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pp. 821–827.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, *17*, 229–264.

de Jonge, F., Roos, N., & Witteveen, C. (2009). Primary and secondary diagnosis of multi-agent plan execution. *Journal of Autonomous Agent and Multiagent Systems*, *18*(2), 267–294.

Eiter, T., Erdem, E., Faber, W., & Senko, J. (2007). A logic-based approach to finding explanations for discrepancies in optimistic plan execution. *Fundamenta Informaticae*, *79*(1-2), 25–69.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*, 61–124.

Grastien, A., Anbulagan, Rintanen, J., & Kelareva, E. (2007). Diagnosis of discrete-event systems using satisfiability algorithms. In *Proc. of Association for the Advancement of Artificial Intelligence (AAAI'07)*, pp. 305–310.

Grastien, A., Haslum, P., & Thiébaux, S. (2012). Conflict-based diagnosis of discrete event systems: Theory and practice. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pp. 489–499.

Gupta, S., Roos, N., Witteveen, C., Price, B., & de Kleer, J. (2012). Exploiting shared resource dependencies in spectrum based plan diagnosis. In *Proc. of Association for the Advancement of Artificial Intelligence (AAAI'12)*, pp. 2425 – 2426.

Heger, F. W., Hiatt, L. M., Sellner, B., Simmons, R., & Singh, S. (2005). Results in Sliding Autonomy for Multi-Robot Spatial Assembly. In *Proc. International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, *173*(5-6), 503–535.

Jonsson, P., & Bäckström, C. (1998). State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, *100*(1-2), 125–176.

Kalech, M. (2012). Diagnosis of coordination failures: A matrix-based approach. *Journal of Autonomous Agents and Multiagent Systems*, *24*(1), 69–103.

Kalech, M., & Kaminka, G. A. (2003). On the design of social diagnosis algorithms for multi-agent teams. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI03)*, pp. 370–375.

Kalech, M., & Kaminka, G. A. (2005). Diagnosing a team of agents: Scaling up. In *Proc. of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'05)*, pp. 249–255.

Kalech, M., & Kaminka, G. A. (2007). On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence*, *171*(8-9), 491–513.

Kalech, M., & Kaminka, G. A. (2011). Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, *27*(3), 393–421.

Kalech, M., Kaminka, G. A., Meisels, A., & Elmaliach, Y. (2006). Diagnosis of multi-robot coordination failures using distributed CSP algorithms. In *Proc. of Association for the Advancement of Artificial Intelligence (AAAI'06)*, pp. 970–975.

Kaminka, G. A., & Tambe, M. (2000). Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, *12*, 105–147.

Lamperti, G., & Zanella, M. (2002). Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, *137*(1-2), 91–163.

Mi, P., & Scacchi, W. (1993). Articulation: an integrated approach to the diagnosis, re-planning, and rescheduling of software process failures. In *Proc. of Knowledge-Based Software Engineering Conference*, pp. 77–84.

Micalizio, R. (2013). Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, *29*(2), 233–280.

Micalizio, R., & Torasso, P. (2007a). On-line monitoring of plan execution: a distributed approach. *Knowledge-Based Systems*, *20*(2), 134–142.

Micalizio, R., & Torasso, P. (2007b). Plan diagnosis and agent diagnosis in multi-agent systems. In *Proc. Congress of the Italian Association for Artificial Intelligence (AI*IA'07)*, Vol. 4733 of *LNCS*, pp. 434–446.

Micalizio, R., & Torasso, P. (2008). Monitoring the execution of a multi-agent plan: Dealing with partial observability. In *Proc. of European Conference on Artificial Intelligence (ECAI'08)*, pp. 408–412.

Micalizio, R., & Torasso, P. (2009). Agent cooperation for monitoring and diagnosing a MAP. In *Proc. of Multiagent System Technologies (MATES'09)*, Vol. 5774 of *LNCS*, pp. 66–78.

Micalizio, R., Torasso, P., & Torta, G. (2006). On-line monitoring and diagnosis of a team of service robots: a model-based approach. *AI Communications*, *19*(4), 313–349.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, *12*, 271–315.

Pencolé, Y., & Cordier, M. (2005). A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, *164*, 121–170.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32 (1)*, 57–96.

Roos, N., & Witteveen, C. (2009). Models and methods for plan diagnosis. *Journal of Autonomous Agent and Multiagent Systems*, *19*(1), 30–52.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1995). Diagnosability of discrete event systems.. *IEEE Transactions on Automatic Control*, *40*(9), 1555–1575.

Sellner, B., Heger, F., Hiatt, L., Simmons, R., & Singh, S. (2006). Coordinated multi-agent teams and sliding autonomy for large-scale assembly. *IEEE - Special Issue on Multi-Robot Systems*, *94*(7), 1425 – 1444.

Steinbauer, G., & Wotawa, F. (2008). Enhancing plan execution in dynamic domains using model-based reasoning. In *Intelligent Robotics and Applications, First International Conference, (ICIRA'08)*, Vol. 5314 of *LNAI*, pp. 510–519.

Tambe, M. (1998). Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, *12*(2-3), 189–210.

Torasso, P., & Torta, G. (2003). Computing minimum-cardinality diagnoses using OBDDs. In *German Conference on AI (KI'03)*, Vol. 2821 of *LNCS*, pp. 224–238.

Torasso, P., & Torta, G. (2006). Model-based diagnosis through OBDD compilation: A complexity analysis. In *Reasoning, Action and Interaction in AI Theories and Systems*, Vol. 4155 of *LNCS*, pp. 280–298.

Torta, G., & Torasso, P. (2007). On the role of modeling causal independence for system model compilation with OBDDs. *AI Communications*, *20*(1), 17–26.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, *15*(4), 27–61.

Witteveen, C., Roos, N., van der Krogt, R., & de Weerdt, M. (2005). Diagnosis of single and multi-agent plans. In *Proc. of International Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pp. 805–812.

Yan, Y., Dague, P., Pencolé, Y., & Cordier, M.-O. (2009). A model-based approach for diagnosing fault in web service processes. *Journal of Web Service Research.*, *6*(1), 87–110.