Supervision and Diagnosis of Joint Actions in Multi-Agent Plans

(Short Paper)

Roberto Micalizio Pietro Torasso
Università di Torino, Dipartimento di Informatica
corso Svizzera 185
Torino, Italy
{micalizio,torasso}@di.unito.it

ABSTRACT

The paper formalizes a distributed approach to the problem of supervising the execution of a multi-agent plan where (possibly joint) actions are executed concurrently by a team of cooperating agents in a partially observable environment. The notions of plan and agent diagnosis are introduced and discussed.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

Keywords

Multi-Agent Planning, Plan execution monitoring, Plan diagnosis

1. INTRODUCTION

Thanks to the recent technological advances many complex tasks can now be solved in a distributed way by means of a Multi-Agent System (MAS). The basic idea consists in decomposing a complex goal into sub-goals, each of which is assigned to an agent (either software or robotic) of a team: the agents of the team cooperate to reach a common global goal. However, as pointed out in [3], a MAS represents an effective solution in distributed problem solving only when the interactions involve just few agents and when the agents do not have to interact heavily.

In order to avoid (or at least to limit) the occurrence of harmful interactions while the agents accomplish their tasks, the agents' activities can be organized in a multi-agent plan (MAP). While a number of approaches to the synthesis of MAPs have been proposed (see e.g., [2, 5]), the synthesis of a MAP is just the first step as the actual execution of a plan may be threatened [1] by the occurrence of unexpected events (e.g., faults in the functionalities of the agents); therefore the execution of the MAP needs to be supervised to detect anomalous situations and to recover from them.

Some Model-Based solutions for supervising (monitoring and diagnosing) the execution of a MAP have been recently proposed (see e.g., [10, 6, 7, 8]). These approaches, however, are unable to deal with *joint actions* (i.e., actions which require the cooperation of more agents to achieve a goal that a single agent could not achieve). In many real cases, joint actions play a relevant role for accomplishing a given task; see for example the construction task scenario addressed in [9] where a number of robots cooperate for assembling habitats on the surface of Mars (or Moon).

Cite as: Supervision and Diagnosis of Joint Actions in Multi-Agent Plans. (Short Paper), R. Micalizio and P. Torasso, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1375-1378.

Dealing with joint actions presents a number of issues: first of all, the agents need to synchronize themselves before starting the execution of a joint action. Moreover, the presence of joint actions introduce further dependencies among the agents: a flaw in an agent could affect other agents which are cooperating with it during the execution of a joint action. This means that novel methodologies for the supervision and the recovery of a MAP must be devised.

In this paper we propose a framework for the supervision of the execution of a MAP: such a framework is sufficiently general to deal with both simple actions executed by a single agent and joint actions involving a number of cooperating agents. In particular, the paper focuses on the problems of detecting as soon as possible the occurrence of anomalies in the execution of a MAP (e.g., action failures) and of diagnosing these anomalies, i.e., providing a set of possible explanations for these failures. Particular attention is devoted to the diagnostic task, which has to highlight not only the possible explanations for the detected failures but even how these failures threaten the execution of the given MAP.

The paper is organized as follows: section 2 introduces the model of a multi-agent plan (MAP), while section 3 addresses the main issues related to the distributed execution of a MAP. In sections 4 and 5 we address the problems of monitoring and diagnosing the execution of the MAP in a distributed way; finally in section 6 we make some concluding remarks.

2. MODELING THE MULTI-AGENT PLAN.

In this paper we focus on a specific class of MAS where the agents in a team $\mathcal{T}$ work together to achieve a common goal G. Since agents cooperate by exchanging services or by executing joint actions, there exist causal dependencies among the activities they perform. In order to model the MAS in a way which highlights both the agents activities and the causal dependencies existing among them, we adopt the notion of Multi-Agent Plan (MAP).

Global plan. The notion of MAP has been formalized by Cox et al. in [4]. Briefly, given a team $\mathcal{T}$ of agents, the MAP is the tuple $\langle A, E, CL, CC, NC\rangle$ such that: A is the set of the action instances the agents have to execute; each action a is assigned to a specific agent i of the team $\mathcal{T}$ and it is modeled in terms of preconditions and direct effects; E is a set of precedence links between actions; CL is a set of causal links of the form $l : a \xrightarrow{q} a'$; the link l states that the action a provides the action a' with the service q, where q is an atom occurring in the preconditions of a'; finally, CC and NC are respectively the *concurrency* and *non-concurrency* symmetric relations over the action instances in A; in particular, the pair $\langle a, a'\rangle$ in CC models a joint action whereas constraints in NC prevent the conflicts for accessing the resources; this is equivalent to the *concurrency requirement* introduced in [10].

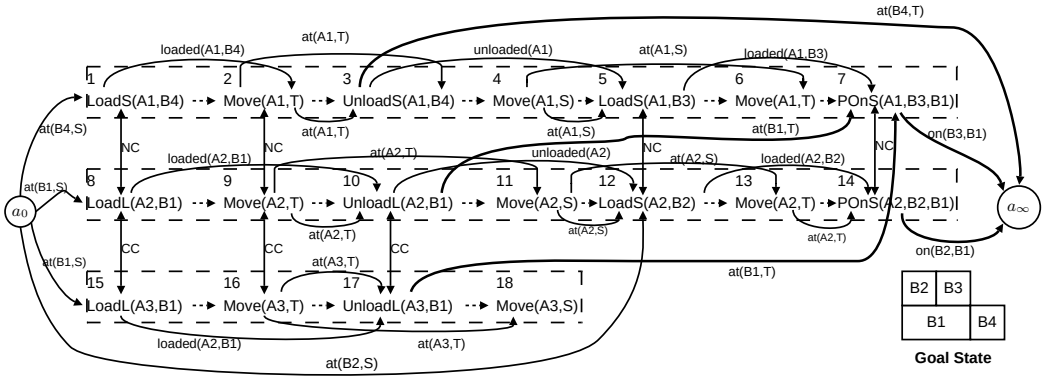

Figure 1: The MAP P to be supervised.

3. DISTRIBUTED PLAN EXECUTION

During the distributed execution of the MAP the agents need to coordinate their activities to prevent the violation of the constraints defined during the planning phase. We adopt a distributed supervision approach (similar to the one discussed in [10, 7]) where a MAP P is decomposed in a number of sub-plans P_i and each sub-plan P_i is assigned to the agent i of the team; each agent performs a local control loop on the progress of the actions it has to execute and communicate with other agents when required.

Local Plans. The decomposition can be easily done by selecting from P all the actions an agent i has to execute. Formally, the sub-plan for agent i is the tuple P_i=$\langle$ A_i, E_i, CL_i, CC_i,, NC_i T_i^{in}, T_i^{out} $\rangle$ where: A_i, E_i, CL_i, CC_i and NC_i are the same as in P restricted to the actions the agent i has to execute; while T_i^{in} (T_i^{out}) is a set of incoming (outgoing) causal links $a \xrightarrow{q} a'$ where a' (a) belongs to A_i and a (a') is assigned to another agent j in the team.

Coordination during plan execution. As concerns the plan execution, the time is assumed to be a discrete sequence of instants. Actions are not scheduled in a rigid timetable as we assume that an action is executed as soon as its preconditions are satisfied. Although actions are modeled in terms of preconditions and effects, their execution may require more than one instant.

When an action is completed at time t, the agent i receives a set of observations obs_t^i relevant for the status of agent i itself. Although in general the observations obs_t^i are not sufficient for precisely inferring the status of agent i, we assume that they are sufficient to evaluate the outcome (*succeeded* or *failure*) of the last action the agent i has executed. Coordination among agents can be achieved in an efficient way by exploiting the causal links and the concurrency or non-concurrency constraints defined in the MAP and maintained by each agent in the definition of its own sub-plan. In the nominal plan execution, coordination is required in three cases. First, when an agent i has to provide a service q to another agent j, i has to inform j when the service q has been provided. Technically, the information about the service provided by an agent to another one is encoded in the causal link l in the MAP P.

Coordination is also needed during the execution of a joint action as it involves the synchronization of two (or more) agents; the set of concurrency constraints CC defined in the MAP P denotes which agents need to cooperate (and when). Finally, explicit coordination is required for executing actions bounded by non-concurrency constraints: in this case coordination is ruled by the set of non-concurrency constraints NC in P and prevents the simultaneous execution of the constrained actions.

Coordination among agents is also needed in case of an action failure; in fact an agent must notify other agents when a service will not be provided as a consequence of a failure.

Running Example. For illustrative purpose we will use a simple example from the blocks world. Let us consider three agents $A1$, $A2$ and $A3$, that cooperate to move the block `B1`, `B2`, `B3` and `B4` in the target location `T` where `B2` and `B3` have to be put on the top of `B1`. Initially the four blocks are in the source location `S`. We distinguish between *small* and *large* blocks. In its nominal behavior an agent can handle one small block, whereas for handling a large block two agents have to join their efforts.

The operations within the source and target locations are constrained as, at each time instant, only one block can be loaded/unloaded in the same location; we assume that the given MAP P does not violate such a constraint.

Figure 1 shows a possible instance of a MAP which achieves the target configuration of blocks. The agents `A2` and `A3` cooperate for loading the large block `B1` (actions `LoadL(A2,B1)` and `LoadL(A3,B1)`), moving it to the target position `T` (actions `Move(A2,T)` and `Move(A3,T)`) and position the block in `T` (actions `UnloadL(A2,B1)` and `UnloadL(A2,B1)`). The agent `A2` has also the task of moving the small block `B2` and putting it on the top of `B1` (action `POnS(A2,B2,B1)`). The task assigned to `A1` involves the transfer of small blocks `B4` and `B3`.

The plan is a DAG whose nodes correspond to actions and edges can be precedence links (dashed), causal links (solid) or concurrency and non-concurrency constraints (solid bidirectional edges labeled with CC and NC respectively). It is easy to see that agents `A2` and `A3` execute some joint actions (e.g., $\langle 8, 15 \rangle$) to move the large block `B1` in its final position. Causal links are labeled with the services an action provides to another one: for example, the causal link from action 1 to action 3 is labeled with the service `loaded(A1,B1)`. The dashed rectangles specify which actions are included in the sub-plans assigned to the three agents.

4. MONITORING THE MAP EXECUTION

The supervision task, that each agent i performs over the execution of action a_t^i it is responsible for, provides two important services:

1) infers the agent belief state $\mathcal{B}_{t+1}^i$ after the execution of a_t^i;
2) assesses of the outcome of the executed action a_t^i.

Agent status. Intuitively, the agent status can be expressed as a set VAR^i of the variables concerning both the status of the agent and the status of the system resources RES. A critical role is played by the subset $HEALTH^i$ of VAR^i which denotes the set of variables concerning the health status of i's functionalities: for each agent functionality f, a variable $v_f \in HEALTH^i$ represents the health status of f; the domain of variable v_f is the set $\{ok, ab_1, \ldots, ab_n\}$ where ok denotes the nominal mode while $ab_1, \ldots, ab_n$ denote non nominal modes.

Agent belief state. since the system is just partially observable,

the agent i observes just a subset of its status variables; in particular, the the variables in $HEALTH^i$ are not directly observable and their actual value can be just inferred. Thus, the set obs_t^i of observations conveys information about a subset of variables in $VAR^i \setminus HEALTH^i$; and in most cases the agent i can determine just a set of alternative states which are consistent with obs_t^i; in literature this set is known as *belief state* and will be denoted as $\mathcal{B}_t^i$.

Action models. As discussed in [5], the model $\Delta(a_t^i)$ of a *simple action* a_t^i (assigned to agent i at time t) is characterized by three parts: a set *var*$(a^i) \subseteq VAR^i$ of *state variables*, a set *pre*(a^i) of *preconditions* and a set *eff*(a^i) of *effects*, where both preconditions and effects are constraints defined over the set *var*(a^i). Since an action may have non deterministic effects, the action model $\Delta(a_t^i)$ can be seen as transition relation where each tuple $d \in \Delta(a_t^i)$ models a possible change in the status of agent i, which may occur while i is executing a_t^i. Each tuple d has the form $d = \langle s_t, s_{t+1} \rangle$ where s_t and s_{t+1} represent two agent states at time t and $t+1$ respectively; each state is an assignment of values to the status variables in *var*(a^i) (the variables in $VAR^i \setminus$ *var*(a^i) are assumed to be constant).

Given the action a^i, *healthVar*(a^i)=$HEALTH^i \cap$*var*(a^i) denotes the set of health status variables, i.e. functionalities, which directly affect the outcome of action a^i. The *healthy formula* for a^i is defined on this set of variables and denotes under what conditions the action behaves nominally and all the expected effects are reached. Formally the set of nominal effects of a^i is:

nominalEff(a^i)=$\{q \in$ *eff*$(a^i) \mid$ *pre*$(a^i) \cup$ *healthy*$(a^i) \vdash q\}$.

On the contrary, when the healthy formula does not hold, the behavior of the action may be non deterministic and some of the expected effects may be missing.

Joint actions. The notion of simple action can be extended to cover the notion of joint action: in fact, as discussed in [5], a joint action can be seen as the simultaneous execution of a subset of simple actions. In this paper we consider a stronger notion of joint action: two simple actions a^i and a^j are part of a joint action $a^{i,j}$ not only because they are executed at the same time, but also because they actively cooperate to reach an effect. This stronger notion of joint action can be captured by exploiting the notion of *dependency set* introduced in [7]. Intuitively, a dependency set highlights the subset of agents among which a strict cooperation is required in a specific time instant.

The notion of dependency set is sufficiently general to cover both simple and joint actions. In fact, when an agent i executes a simple action a_t^i, the agent i is the only member of its dependency set $I(t)$; on the other hand, when agent i executes a joint action $a_t^{i,j}$, the dependency set $I(t)$ associated with i at time t contains both agent i and agent j. Therefore the notation $a_t^{I(t)}$ can denote either a simple or a joint action according to the number of agents involved in the dependency set $I(t)$.

The state estimation process. Let $a_t^{I(t)}$ denote the (joint) action executed by the agent(s) in the dependency set $I(t)$ (for the sake of readability we will write a_t^I whenever the time of the dependency set is obvious from the context); the process for estimating the (joint) belief state after the execution of an action a_t^I can be formalized in terms of Relational Algebra operators (see [7] for details). In particular, given agent i and its dependency set $I(t)$ at time t, let $\mathcal{B}_t^I$ be the joint belief state of agent i, let $\Delta(a_t^I)$ the model of the (joint) action which the agent i has to execute at time t, the joint belief state at time $t+1$ is:

$\mathcal{B}_{t+1}^I = \text{PROJECTION}_{VAR_{t+1}^I}(\text{SELECTION}_{obs_{t+1}^I}(\mathcal{B}_t^I \text{ JOIN} \Delta(a_t^I)))$.

The join operation $\mathcal{B}_t^I \text{ JOIN} \Delta(a_t^I)$ represents the prediction step as it estimates the set of possible states of the agents in I at time $t+1$. However, the set of predictions resulting from the join operation is in general spurious as it predicts all possible evolutions: we have to take into consideration the observations received by each agent $i \in I$ at time $t+1$ to restrict the set of possible states. The selection operation $\text{SELECTION}_{obs_{t+1}^I}$ has the effect of pruning off all those predictions which are inconsistent with the agent observations; intuitively, $obs_{t+1}^I = \bigcup_{i \in I} obs_{t+1}^i$. Finally, the belief state at time $t+1$ is obtained by projecting the resulting estimates over the status variables of the agents at time $t+1$ i.e., over the set of variables VAR_{t+1}^I.

5. DIAGNOSING A MAP

Action outcome. The outcome of action a_t^I is either *succeeded* or *failed*; in particular, action a_t^I is considered succeeded when all its nominal effects *nominalEff*(a_t^I) have been achieved after its execution. When a_t^I is a joint action, the agents in I share a jointed belief state $\mathcal{B}_t^I$ resulting from the conjunction of the belief states of the agents in $I(t)$. In order to be conservative, we consider action a_t^I successfully completed only when the nominal effects of a_t^I hold in every possible state in $\mathcal{B}_t^I$. Of course, when we can not assert that action a_t^I is succeeded we assume that the action is failed.

Agent diagnosis. Whenever the outcome of a_t^I is failed, a diagnostic process is activated in order to infer a set of possible explanations for such a failure; i.e. which combinations of faults in the agents in I may be the cause of the failure of action a_t^I.

In the relational framework we propose, given the failure of action a_t^I, the agent diagnosis D_t^I can be determined simply by projecting the joint belief state over the health status variables, formally:

$D_t^I = \text{PROJECTION}_{healthVar(a_t^I)}(\mathcal{B}_t^i)$.

Each tuple $d \in D_t^I$ is an assignment of values to the variables in *healthVar*(a_t^I), moreover, every assignment d is consistent with the observations obs_t^I; hence, every tuple d is a possible explanation for the failure of action a_t^I.

It is worth noting that, as a consequence of the partial observability, the agent diagnosis is in general ambiguous (i.e., it maintains a number of alternative explanations).

Plan diagnosis. While the agent diagnosis singles out the not nominal health status of the agents, the plan diagnosis aims at discovering which other actions in the plan could be indirectly affected by the failure of action a_t^I. Two different kinds of threats have to be considered: the *causal threats* and *fault threats*.

Fault threats. For the sake of simplicity in the discussion let's consider first fault threats. The agent diagnosis D_t^I allows one to determine which actions may be threatened by the not nominal health status of the agents in I. Intuitively, given an agent $i \in I$, the action $a^i \in A_i$ is threatened by the agent diagnosis D_t^I when:

- the action a^i has still to be execute (i.e., $a_t^i \prec a^i$) *and*
- the action requires at least one functionality f which is assumed to be faulty in D_t^I.

More formally, the action a^i is threatened by the diagnosis D_t^I when the healthy formula *healthy*(a^i) is inconsistent with D_t^I. In the following we will denote as $fThreatenedActs(a_t^I, i)$ the set of actions agent i has still to execute and which are threatened by the agent diagnosis.

Causal threats. The actions threatened through causal links can be determined by considering both the failed action a_t^I and the set of actions threatened by the agent diagnosis. Intuitively, an action a is threatened through a causal links $l : a' \xrightarrow{q} a$ when it is no longer guaranteed that the action a' provides the service q; this may happen either because a' is failed or because a' is in turn threatened.

In the following we denote as $cThreatenedActs(a_t^I)$ the set of actions threatened by a causal links; this set can be determined just by propagating the failure through the existing causal links. It is worth

noting that, while the actions threatened by the agent diagnosis belong to same agent i, the actions threatened through causal links may have been assigned for execution to any other agent; i.e., the causal threats may have a much wider impact than the fault threats. The plan diagnosis can be therefore represented as the union of the two sets of threatened actions:
$thrActs(a_t^I) = fThreatenedActs(a_t^I) \cup cThreatenedActs(a_t^I)$.
Namely, the plan diagnosis consists of all the actions which are threatened by the failure of action a_t^I.

Missing goals. A *missing goal* is a service agent i is responsible for, which can not be provided as a consequence of the failure of action a_t^I (where i belongs to the dependency set I). In principle, it would be sufficient to provide all the missing goals in order to reach the MAP's goal despite the occurrence of the action failure. To formally characterize the concept of missing goal we introduce the notion of *primary effect* as the nominal effect q of an action a ($q \in$ *nominalEff*(a)) such that either q is a an atom which appears in the global goal G or q is a service that an agent provides to another one. In general, given an action a, *primary*(a) denotes the (possibly empty) set of primary effects provided by a.
Given the plan diagnosis $thrActs(a_t^I)$, $thrActs(a_t^I, i)$ denotes the subset of threatened actions in the plan assigned to agent i; formally $thrActs(a_t^I, i)$=$\{$actions $a \in thrActs(a_t^I) | a \in A_i\}$. The set of missing goals for agent i is the set of goals that agent i can no longer achieve:
$missingGoals(i)$=$\bigcup_{a \in thrActs(a_t^I,i)} primary(a)$. The notion of missing goals can be easily extended to all agents in the dependency set I: $missingGoals(I)$=$\bigcup_{i \in I} missingGoals(i)$.

Running Example. To make clear the diagnostic process let's reconsider the previous example and assume that the joint action $\langle 9, 16\rangle$ (executed by agents A2 and A3) fails. In particular, $\langle 9, 16\rangle$ requires that both agent A2 and agent A3 execute a move action from the source location S to the target location T. Since the available observations after the action execution provide the information that the two agents have not reached the target location T, the agents infer that the outcome of joint action is failed because the nominal effects have not been achieved. For explaining this action failure the agent diagnosis $D^{A2,A3}$ is inferred and in particular it includes the following alternative explanations:

	power(A2)	*mobility(A2)*	*power(A3)*	*mobility(A3)*
1	*reduced*	*ok*	*full*	*ok*
2	*full*	*ok*	*reduced*	*ok*

It follows that every action assigned to agent A2 (A3) which requires power *full* is threatened by the agent diagnosis. Let us assume that, apart from other functionalities, only the joint actions require that the power is full to be successfully completed (in other words, power full is a conjunct of the healthy formula). Therefore the set $fThreatenedActs(\langle 9, 16\rangle)$ includes the joint action $\langle 10, 17\rangle$. Hence the set of actions threatened through the causal links is $cThreatenedActs(\langle 9, 16\rangle)$= {7,10,11,12,13,14,17,18}, involving the actions of agent A1 too.
However, in order to determine the missing goals we have to consider the primary effects of the actions, assigned either to A2 or to A3, which are threatened by the failure. In this example, the actions which satisfy these conditions are actions $\langle 10, 17\rangle$ and 14. Thus, the set of missing goals includes the effects of these two actions: at(B1,T) and on(B2,B1).

Implementation and preliminary results We have implemented this approach by extending the implementation discussed in [7] in order to handle joint actions. The relations representing both the belief states and the action models have been encoded by means the Ordered Binary Decision Diagrams (OBDDs) and the relational operations have been mapped into standard operations on OBDDs. A prototype has been implemented in Java JDK 1.5 and exploits the JavaBDD[1] package for symbolically encoding and manipulating OBDDs. The robotic agents are simulated in a software environment and are implemented as threads running on the same PC[2]. The preliminary results collected so far are encouraging; in fact, at each time instant, the plan supervision (monitoring, agent diagnosis and failure propagation) performed by each agent requires on average 5 msec. and the maximum absolute CPU time is 30 msec.

6. DISCUSSION AND CONCLUSIONS.

In this paper we have pointed out the interplay between plan and agent diagnosis in estimating the impact of an action failure: in particular, for singling out what sub-goals are no more guaranteed to be achieved by the current MAP after the occurrence of an action failure. The set of *missingGoals* is the starting point of any strategy aimed at recovering the execution of the MAP as it represents the set of services that need to be provided in an alternative way (typically a replanning step is needed). The proposed framework is sufficiently general for monitoring and diagnosing the execution of joint actions, which require tight cooperation among the agents not only during the execution but also during the supervision process.

Recently, Roos et al. [10] have proposed a distributed approach to the diagnosis of a MAP, which is relevant as it introduces the notion of plan diagnosis. Our framework extends the approach by Roos et al. since the non deterministic effects of failures are explicitly represented. As a consequence we can complement the plan diagnosis with the notion of agent diagnosis.

The preliminary experimental results are encouraging and pave the way for testing the approach in more challenging domains such as the air traffic control domain and the space exploration scenario.

7. REFERENCES

[1] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *Proc. AAAI90*, pages 318–323, 1990.

[2] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *JAIR*, 14:105–136, 2001.

[3] N. Carver and V. Lesser. Domain monotonicity and the performance of local solutions strategies for cdps-based distributed sensor interpretation and distributed diagnosis. *Journal of AAMAS*, 6:35–76, 2003.

[4] J. S. Cox, E. H. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *Proc. AAMAS05*, pages 821–827, 2005.

[5] R. M. Jensen and M. M. Veloso. Obdd-based universal planning for synchronized agents in non-deterministic domains. *JAIR*, 13:189–226, 2000.

[6] M. Kalech and G. Kaminka. Diagnosing a team of agents: Scaling up. In *Proc. AAMAS05*, pages 249–255, 2005.

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

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

[9] B. Sellner and R. Simmons. Towards proactive replanning for multi-robot teams. In *Proc. Int. Work. PS in Space*, 2006.

[10] C. Wittenven, N. Roos, R. van der Krogt, and M. de Weerdt. Diagnosis of single and multi-agent plans. In *Proc. AAMAS05*, pages 805–812, 2005.

[1] http://sourceforge.net/projects/javabdd

[2] Intel Pentium 1.86 GHz, RAM 1 GB, Windows XP OS.