On-line monitoring and diagnosis of a team of service robots: A model-based approach

Roberto Micalizio, Pietro Torasso and Gianluca Torta
Dipartimento di Informatica, Università di Torino, Italy
E-mail: {micalizio,torasso,torta}@di.unito.it

Abstract. The paper presents an approach for the on-line monitoring and diagnosis of multi-robot systems where services are provided by a team of robots and the environment is only partially observable via a net of fixed sensors. This kind of systems exhibits complex dynamics where weakly predictable interactions among robots may occur. To face this problem, a model-based approach is adopted: in particular, the paper discusses how to build a system model by aggregating a convenient set of basic system components, which are modeled via communicating automata. Since the dynamics of a multi-robot system depend on the actions performed by the robots (and actions change over time), the global system model is partitioned into a number of submodels, each one describing the dynamics of a single action.

The paper introduces the architecture of the Supervisor which has to track the actions progress and to infer an explanation when an action is completed with delay or fails. The Supervisor includes two main modules: the On-line Monitoring Module (OMM) tracks the status of the system by exploiting the (partial) observations provided by sensors and robots. When the monitor detects failures in the actions execution, the Diagnostic Interpretation Module (DIM) is triggered for explaining the failure in terms of faults in the robots and/or troublesome interactions among them.

The RoboCare domain has been selected as a test bed of the approach. The paper discusses experimental results collected in such a domain with particular focus on the competence and the efficiency of both the OMM and the DIM.

Keywords: Model-based diagnosis, intelligent monitoring, multi-agent systems, action execution

1. Introduction

In the recent years there has been a growing interest in integrating planning, scheduling, monitoring and diagnosis for controlling and supervising complex autonomous systems. Pioneering projects such as the Remote Agent Experiment [13] have shown that monitoring and diagnosis play a central role since robust planning depends on the assessment of the status of the system, including failures.

Recent advances in the fields of cognitive robotics and multi-robot systems have paved the way for approaching complex tasks by means of distribution of subtasks among robotic agents and cooperation among them (see e.g., the RobotCup competition [1] and the field of open collective robotics [9]).

Informally, for supervising a dynamic system we have to estimate the status of the system at each time instant and, whenever the observed system behavior deviates from the nominal, expected behavior, we have to infer a set of possible explanations about the causes of the anomalous behavior. In some approaches monitoring and diagnosis are performed in a distributed way by a team of software agents (see e.g., [10] and [15]). In such multi-agent approaches the agents have to be able to compute local diagnoses i.e., diagnoses for just a portion of the system; in order to aggregate a global diagnosis, the agents have to communicate with each other so that the problem of communication overhead can often arise.

This paper addresses the problems of on-line monitoring and diagnosing a multi-robot system. We conceive the activity carried on by the set of robots as the concurrent execution of a plan in a complex and dynamic environment. For simplicity we will consider that the plan executors are robots; however their nature is domain dependent, for example in the Air Traffic Control domain (see [8]) plan executors are the airplanes which execute their own flight plan.

More precisely, the paper addresses the on-line monitoring problem by introducing a centralized Supervisor which provides an external module, such as a Planner, with explanations of the anomalous behavior of the system (i.e., diagnoses) by exploiting a qualitative model of the system.
As stated above, some approaches to multi-agent diagnosis assume that the agents are able to perform self-diagnosis (see, for example, the social diagnosis introduced in [10]). However, this may be a strong requirement in many domains. In our discussion we release this requirement and assume that robots are not necessarily able to perform self-diagnosis, although they can provide the Supervisor with useful pieces of information, for example about their position. We also allow the robots to have some degree of autonomy in particular as concerns basic skills such as navigation, impact avoidance and negotiation with other robots; finally, we assume that the environment is partially observable by means of a net of fixed sensors.

As suggested in [3], each action in the monitored plan is associated with a time deadline which avoids the Supervisor to wait indefinitely for the completion of that action, i.e., the action deadline defines the time by which the action goal has to be achieved. In our approach we model the duration of the actions with a tolerance interval; for this reason we associated each action with two deadlines: the first deadline models the nominal duration of the action, the second deadline models the maximum delay within which the action has to be successfully completed. An action which is not completed when the second deadline expires is considered failed.

Monitoring and diagnosing multi-robot systems presents a number of challenges. A first issue concerns the on-line monitoring task. In classical planning problems, the actions are considered atomic and are modeled by taking into consideration only their preconditions and post-conditions. In [16] an approach to plan monitoring and diagnosis is discussed; the approach models atomic actions as functions from preconditions to post-conditions; moreover, the authors impose that the plan to be monitored satisfies a concurrency requirement, i.e., two concurrent actions can not require the same resource.

Since in our approach we consider also non-atomic actions the Supervisor needs more detailed actions models. More precisely, the model of an action must define not only the initial and the final states (where the preconditions and post-conditions respectively hold), but also one or more possible paths touching states where intermediate conditions must hold.

Moreover we relax the concurrency requirement; thereby, during the plan execution, two (or more) concurrent actions can request the same resources. Relaxing the concurrency requirement has the effect of introducing competitions among robots for accessing the resources. As we will see, these robot interactions are threats (see [3]) for the successful completion of the plan and represent a challenging issue that the diagnosis has to face. As discussed in [3], a Planner coping with any particular class of domain dependent threats has a choice of either (1) attempting to prevent the threats (as in [22]), or (2) attempting to deal with threats individually as they arise. In this paper we are interested in the second of these two choices; in fact, the main idea is to provide the Planner with a set of possible explanations about an observed action failure; these explanations can be exploited by the Planner for adjusting on-line the plan and overcoming (if possible) the causes of the failure.

Even if the model of an action is detailed enough, in general it is not deterministic since it has to take into account multiple alternative ways of reaching the action post-conditions and, more important, it has to deal with the cases where the action can not be carried out, e.g., due to the occurrence of one or more faults in the functionalities of the robot performing the action.

Since the system is only partially observable and the action models are non-deterministic, the Supervisor can not unambiguously determine the current status of the system at each time instant; instead, it needs to keep track of a set of alternative system states which are consistent with the received observations; this set is known in the literature as belief state. In principle a belief state could be exploited by other modules (e.g., a Planner) or by a human supervisor during the decision process which aims to overcome (if possible) the root causes of a detected anomaly. However, as we will see, the number of possible states within a single belief state can be very large. In addition, a belief state represents the system states at a level that may be too low to figure out what has gone wrong and why. In order to provide useful information, the Supervisor has to infer a high-level interpretation of the belief state (actually, of a history of belief states) in order to make evident the causes of an action failure.

This high-level interpretation task is not trivial; in fact, another challenging aspect of a multi-robot system is that an action can fail not only because of a fault occurring in the robot performing it, but also as a consequence of undesired interactions among robots that arise when the robots compete for accessing a resource. These interactions are not completely predictable since they are a consequence of the actions currently performed by the robots.

In the present paper we adopt a model-based approach: we model the basic entities of the system by...
means of the communicating automata formalism introduced in [4] (similarly to what has been done for the diagnosis of distributed systems in [14]). However the diagnosis of multi-robot systems is inherently different from the one of distributed systems. Since the possible interactions among system entities are not known in advance and change over time, we introduce a method for updating the global system model every time an action is completed or a new action starts; the global model is obtained by dynamically aggregating convenient sets of component models depending on current actions.

As we will see, the global model of the system plays an important role in the monitoring phase. The model is also useful to the Supervisor to infer diagnoses of the overall system behavior; however we shall see that, in order to obtain diagnoses, some additional heuristic information is needed.

The main ideas in the paper will be exemplified in the context of RoboCare [7], an Italian project involving several partners, where a team of mobile robots is intended to provide services useful for the care of the elderly.

The paper is organized as follows. In Section 2 we introduce the RoboCare domain and the overall architecture of the system. In Section 3 we describe how we model the entities involved in the studied systems. Section 4 discusses how the global transition relation of the system is built by means of the composition of the basic entities. In Sections 5, 6 and 7 we describe in detail the On-line Monitoring Module (OMM) and the Diagnostic Interpretation Module (DIM). Finally, in Section 8 we present experimental results collected from a series of tests performed by means of a software simulator of the RoboCare domain, and in Section 9 we discuss some related work and conclude.

2. The RoboCare case study

In RoboCare, services are provided by mobile robotic agents that are autonomous as concerns navigation and negotiations with other robots for accessing resources. Typical actions performed by robotic agents involve bringing objects (such as meals or medications) to the beds of the patients distributed in a number of rooms. For this reason the robots must be able to navigate in the environment and to manipulate objects by means of a robotic arm.

In [7] the general architecture of the system is introduced and three main components are identified: the Planner, the Scheduler and the Supervisor. The Planner synthesizes a partial-order plan (POP) where each action is assigned to a robot. The Scheduler schedules the execution of an action $act$ when all the actions that precede $act$ in the POP have been completed successfully and the preconditions of $act$ are satisfied. Finally, the Supervisor has the task of monitoring the correct execution of the plan.

It is worth noting that the task of the Supervisor is made possible not only via the pieces of information provided by the robots but also, and more important, by the pieces of information flowing from a net of sensors located at fixed positions in the environment. The presence of this net of sensors guarantees a partial observability of what is going on in the environment.

Figure 1 shows the high level architecture of the Supervisor as described in this paper, and the relationships among the Supervisor and the Planner and Scheduler modules.

The On-line Monitoring Module (OMM) of the Supervisor is responsible for checking the progress in the execution of the scheduled actions by assessing system status. Since the Supervisor has not a complete observability of the environment, it has to estimate the set of system states which are consistent with the observations coming from the net of fixed sensors and with pieces of information possibly volunteered by robots themselves. By taking into consideration the expected
duration of the actions that have to be performed by the robots and the assessed status of the system, the OMM detects failures and/or delays in actions execution and in such cases it triggers the Diagnostic Interpretation module (DIM) which has the task of providing (via the Presentation Module) the Planner and Scheduler with an explanation (i.e., diagnosis) of the detected failures in terms of faults in robotic agents and/or occurrence of competition for resources among robots.

3. Modeling the domain

This section provides a description of the approach adopted for modeling the relevant entities of the domain; we have chosen a model-based approach for representing in a formal way the basic entities of a multi-robot system and how these entities evolve. As we will see in Section 4, the models of the basic entities are composed into a complete model of the activities performed by the robots.

3.1. The basic domain entities

The environment where the robots act basically consists of a set of rooms connected together by doors. More specifically two rooms $R_i$ and $R_j$ are considered adjacent if they are connected by at least one door $D_{ij}$.

The environment also involves a set of passive entities used by the robots in carrying out their actions. For example, in the RoboCare domain, besides the doors, the passive entities are beds (for elderly people) and trayracks; in other domains, for example an office-like domain, passive entities could be desks and filing cabinets. Since a team of robots performs many actions concurrently, the same passive object may be requested for use by more than one robot at a time and therefore its use may need to be regulated by imposing constraints (for example a door can be accessed by just a robot at a time); in such a case we treat the passive object as a resource and we model its use via a set of appropriate constraints (see next section).

We assume that the environment is known both to the Supervisor and to the robots, i.e., the general topology of the environment in terms of rooms, doors, and other passive objects is known. However, we cannot assume that the environment is completely accessible to the Supervisor, so we have to deal with partial observability. For this reason the Supervisor maintains just a coarse, qualitative description of the environment based on the subdivision of the space into areas.

Since resources play a critical role in the evaluation of the success or failure of the actions performed by the robots, two special areas are associated with each resource $res$: critical area $res.CA$ denotes the area from which $res$ can be accessed; request area $res.RA$ denotes the area immediately surrounding $res.CA$. From the point of view of the Supervisor the position of a robot is specified in terms of areas: a robot could be located within a critical or request area of a resource, or into the transit area $TA$ of a room (i.e., all the space of a room that is not part of a critical/request area is modeled as a single transit area used by robots to move from one resource to another). While the number of robots within a critical area is limited, we assume that the transit areas and the request areas do not have any limit.

Apart from pieces of information volunteered by the robots, the Supervisor has a partial observability of the environment via a net of fixed sensors. Typically these sensors are associated with resources in order to detect events occurring within the critical and/or the request area of the resource itself. Moreover, the sensors are able to identify which specific robot has caused a detected event; the identity of the robot is included in the message the sensor sends to the Supervisor.

Figure 2 shows a typical RoboCare environment. All the resources (beds, doors and trayrack) are represented with their critical (light green) and request (light blue) areas. The sensors are displayed as black points nearby the resources they monitor. The white space of a room represents the transit area of the room itself.

3.2. Modeling the dynamics of the system

The dynamics of the multi-robot systems we are interested in is determined mainly by the actions cur-
rently assigned to the robots; moreover, the progress
made by a robot in performing its assigned action de-
2pends, among others, on the (absence of) competi-
tion for resources and its health status (occurrence of
faults).
For this reason the system can be suitably described
by combining the models of three main types of enti-
ties:
– Robot Functionalities, i.e., the skills of the robots
and their possible failures.
– Action Templates, i.e., detailed, non-deterministic
schemas of actions execution.
– Resources.

The events that can be actually observed by the Su-
3pervisor are specified by modeling the Robots Self-
Assessment Skills (which determine the situations when
a robot can/must send a message to the Supervisor) and
the Fixed Sensors (which are able to send messages to
the Supervisor according to the captured events).

The behavior of each type of entity is described
by means of a communicating automaton; the global
model of the system can then be obtained via a proper
synchronization of communicating automatons. While it
is well known that the synchronization among two or
more communicating automatons is based on the ex-
change of internal events [17], it is important to distin-
guish specific classes of System events (as we shall see,
these events do not disappear during synchronization):
– System Input Events:
  ♦ Robot Progress Events $\Sigma_{\text{prgEvn}}$: each event in
    $\Sigma_{\text{prgEvn}}$ is triggered by a robot while it is per-
    forming an action and represents a progress in
    the completion of the action.
  ♦ Faults $\Sigma_{\text{faults}}$: set of possible faults of the ro-
bots.
– System Output Events:
  ♦ Observed Messages $\Sigma_{\text{obsMsg}}$: each event in
    $\Sigma_{\text{obsMsg}}$ corresponds to a message sent either by
    a fixed sensor or by a robot to the Supervisor.

Before discussing in detail each type of domain en-
tity, it is worth noting that we assume that the com-
unication between the robots and the Supervisor as well
as between the sensors and the Supervisor is reliable
and instantaneous\(^1\). However, the framework we pro-
pose could be easily extended to model the communi-
cation infrastructure as a set of components (including
their faults).

\(^1\)This assumption is often made, see, for example [14].

\(\text{Robot Functionalities. A robot functionality is a skill of the robot (e.g., a robot with the handling functionality is able to take an object); a functionality is characterized by a set of behavioral modes, one of which is the nominal mode and the others are faulty or degraded modes.}

\(\text{For example the robot mobility in the RoboCare scenario can be in the OK, SLOWDOWN or BROKEN behavioral modes (see Fig. 3).}

\(\text{A set of available capabilities} \Sigma_{\text{cap}} \text{characterizes each behavioral mode of a functionality (e.g., the OK mode of mobility offers the move capability). We model a robot functionality by a communicating automaton where:}

\(\text{– each state represents a behavioral mode,}
\(\text{– arcs between states represent spontaneous evolutions between behavioral modes due to the occurrence of faults,}
\(\text{– if a capability} cap \text{is available in behavioral mode} bm \text{then the state representing} bm \text{has a self-loop which emits an event labeled} cap.

\text{More formally, a robot functionality} \text{fun} \text{is modeled through the automaton} < Q, \Sigma_{\text{faults}}, \Sigma_{\text{out}}^{\text{cap}}, \Sigma_{\text{funFaults}}, E > \text{where:}

\(\text{–} Q \text{is the set of behavioral modes of the functionality} \text{fun},
\(\text{–} E \subseteq (Q \times \Sigma_{\text{faults}} \times 2^{\Sigma_{\text{out}}^{\text{cap}}} \times 2^{\Sigma_{\text{funFaults}}} \times Q) \text{is the set of transitions},
\(\text{–} \Sigma_{\text{cap}} \text{is a set of internal events which are used for the synchronization with other communicat-
ing automatons. In particular the events in} \Sigma_{\text{cap}} \text{represent the capabilities exposed by a functionality}^2\)
\(\text{–} \Sigma_{\text{funFaults}} \text{is another set of internal events which are used to model the ability of a robot to perform auto-diagnosis about its functionality} \text{fun}; \text{in particular, by means of the synchronization with the robot self-assessment automaton (see later) the robot is able to send an auto-diagnosis message triggered by the occurrence of a fault.}

\(^2\)Since the internal events can be used as input or output events we indicate their use in the superscript, e.g., $\Sigma_{\text{out}}^{\text{cap}}$ means that the events in $\Sigma_{\text{cap}}$ are emitted.
Figure 3 shows that the capability move is available in behavioral modes OK and SLOWDOWN but not in BROKEN. Moreover, notice that $f_1$, $f_2$ and $f_3$ are the spontaneous events through which we model the occurrence of a fault; for example when the mobility functionality is in the nominal mode and the $f_1$ event occurs, the mobility evolves in the SLOWDOWN mode.

In the RoboCare domain we model two robot functionalities; the mobility functionality (Fig. 3) that allows a robot to move, and the handling functionality that allows a robot to take (and hold) an object. These two functionalities can be used simultaneously when a robot must bring an object to a patient.

**Action Templates.** Each robot action is defined through an *Action Template* which provides information about the sequence of sub-steps that the robot has to take for completing the action. The Action Template indicates also which *Resources* and which *Robot Functionalities* are required in order to carry out the action. In other words, the Action Template is a detailed model of an action which specifies not only a fine-grained plan, but also restrictions on the availability of resources and of robots capabilities that may hold for the whole duration of the action or in some specific intervals.

In our approach an action is modeled as a communicating automaton such that:

- each state defines a list of constant assignments to the variables concerning the robot status, in particular those status variables whose value is affected by the action;
- each transition is labeled with an exogenous input event in $\Sigma_{\text{prgEvn}}$ corresponding to an action sub-step; moreover each transition consumes internal events in $\Sigma_{\text{cap}}$ emitted by the required functionalities; all these input events could be considered as necessary conditions for a transition from state $s_i$ to state $s_j$;
- each transition is also labeled with one or more internal events in $\Sigma_{\text{prgMsg}}$ emitted when the transition is taken. The events in $\Sigma_{\text{prgMsg}}$ describe a state change that is not directly observable, but may indirectly trigger messages sent by robots or sensors.

More formally the Action Template automaton is defined as $<Q, \Sigma_{\text{prgEvn}}, \Sigma_{\text{cap}}, \Sigma_{\text{prgMsg}}, E>$ where:

- $Q$ is the set of action states,
- $E \subseteq (Q \times \Sigma_{\text{prgEvn}} \times 2^{\Sigma_{\text{cap}}} \times 2^{\Sigma_{\text{prgMsg}}} \times Q)$ is the set of transitions.

Figure 4 shows the automaton for the action *GoToRoom*(k) in the RoboCare scenario. In particular the action requires robot $Rb_k$ to move from its current position in room $R_i$ to the transit area of room $R_j$ through the door $D_h$; in such a case the status variables concerning the position of $Rb_k$ are involved, so the state labels such as $Bed_i.CA$ represent the value assigned to the robot status variable $Rb_k$. It is worth noticing that the automaton is a template, parametric in the arguments of the *GoToRoom*() action and in the initial position of the robot $Rb_k$; different initial states correspond to different initial positions of $Rb_k$. The intermediate states of the automaton represent, in the sample case, intermediate places where $Rb_k$ has to transit in order to reach its final destination $R_i.T.A$. Note that all the transitions consume a move event emitted by the automaton of the mobility functionality (if move is available in the current behavioral mode of the mobility); thus the move capability is essential not only at the beginning but during all the execution of the *GoToRoom* action.

**Resources.** As discussed above, the resources are those passive entities of the environment whose use is limited by constraints. Although a resource is static, the dynamics of its use has to be modeled since the resource may be required by different concurrent actions at the same time. For this reason, we have to introduce the status of the resource $res$ by taking into account that $res$ can be accessed by a limited number of robots per instant (in most cases only one robot per instant) and that $res$ can be requested by several robots at the same time. Obviously there is a strict relation between the use of a resource $res$ and the localization of the robots. In fact a resource $res$ can be accessed by a robot $Rb_k$ when $Rb_k$ is located in the critical area $CA$ of $res$ (that is position $Rb_k.pos = res.CA$); whereas the fact that $Rb_k$ is located in the request area of $res$ (i.e., $res.RA$)
is not sufficient to state that $R_{b_k}$ has requested $res$, since e.g., $R_{b_k}$ could have just left $res.CA$. Therefore, for representing the status of $res$ we should consider all the possible combinations of robots that are trying to access $res$, that have released $res$, and that are using $res$. An efficient way for representing the status of a resource involves the decomposition of the problem by considering each robot separately. More precisely, the model of a resource is a communicating automaton which is duplicated in as many copies as the robots of the team; each of these copies is associated with a robot $R_{b_k}$, and indicates how the status of the resource changes as a consequence of the behavior of $R_{b_k}$. Formally, the communicating automaton of a resource is defined as $< Q, \Sigma_{prgMsg}, \Sigma_{out}, E>$, where:

- $Q$ is the set of the internal states of the resource.
- $E \subseteq (Q \times \Sigma_{prgMsg} \times 2^{\Sigma_{out}} \times Q)$ is the set of transitions which describe how the status of $res$ evolves.
- The events in $\Sigma_{prgMsg}$ are emitted by the action template automaton (i.e., the automaton that describes the action performed by $R_{b_k}$); whereas the events in $\Sigma_{out}$ synchronize with the automaton of the fixed sensor associated with $res$ (if any) which will be discussed later.

Figure 5 shows the automaton of the resource door $D_h$ associated with the robot $R_{b_k}$, in the context of the RoboCare domain. In particular the status of resource $D_h$ is free w.r.t. $R_{b_k}$ when $R_{b_k}$ is not located within the areas of $D_h$; it is requested when $R_{b_k}$ enters into $D_h.AA$; it is busy when $R_{b_k}$ enters into $D_h.CA$; it is released when $R_{b_k}$ leaves $D_h.CA$; and finally it returns to free when $R_{b_k}$ leaves $D_h.AA$. So the automaton defines the steps through which a robot accesses and releases a resource. Since the status of a resource $res$ is decomposed in as many copies as the robots of the team, we have to force consistency among these copies by means of a set of constraints. Thus, the model of a resource consists also of a set of resource constraints that regulate concurrent access to the resource. A detailed discussion about resource constraints is given in Section 4.4.

**Fixed Sensors.** Changes in the world status due to execution of actions can be made observable to the Supervisor through messages sent by fixed sensors.

A fixed sensor can have its internal status, and may fail in the same way the functionalities of robots can fail. However, for the rest of the paper, we assume that a sensor has just two different working modes, the nominal mode in which the sensor has the highest precision (i.e., it is able to detect all possible events occurring in its monitored area), and the degraded mode in which the sensor is able to detect only some events occurring within the monitored area. Moreover, each mode is represented via a different model and we assume that the working mode of the sensor is known and does not change during the time interval we are interested in for monitoring purpose; however, our framework could easily be extended to deal with (fault) mode transitions of the sensors.

A fixed sensor is modeled as a communicating automaton where each state represents an internal state of the sensor. An arc between two states $s_i$ and $s_j$ accepts an event emitted by a copy of a resource automaton and emits an observable event representing the message sent to the Supervisor. Since a sensor automaton synchronizes with a resource automaton, also the automaton of a sensor is duplicated in as many copies as the robots of the team. Each of these copies is associated with a robot so that a sensor automaton is able to detect the events triggered by the associated robot only.

Figure 6 shows the automaton for the door sensor in the RoboCare domain when the sensor is in the nominal mode: all the detected events are made observable to the Supervisor by means of an observable message; in particular the message specifies which event has been detected and which robot has caused the event. Figure 7 shows the same sensor in a degraded mode with loss of precision: in this mode, just some of the incoming events cause the emission of a message to the Supervisor.
Robot Self-Assessment Skills. Our framework easily supports teams of heterogeneous robots. In particular, there may be robots with different abilities concerning the assessment of the status (both of the environment and of their functionalities). For example, there may be robots which are able to perform auto-diagnosis, and some others which are able to recognize particular harmful robots interactions. Obviously, a robot that is able to perform (partial) assessment of the status provides the Supervisor with this information, so that the Supervisor can exploit it.

In the specific RoboCare domain we organize the robots of the team into three categories:

– **Completely Unaware Robots**: these robots are not able to perform any assessment of the status of the environment and of their functionalities; thus they never send a message to the Supervisor.

– **Partially Aware Robots**: each robot $R_b^k$ in this category has two abilities:
  - assesses the presence of another robot in the critical area of a resource it currently needs and sends to the Supervisor the message `UNACCESSIBLE R_b^k res_CA`.
  - detects whether it is holding/releasing an object `obj` (also when the object is held or released as a consequence of a fault in the arm of the robot) and sends to the Supervisor the messages `GRASPING obj` and `NOT-GRASPING obj` respectively.

– **Fully Aware Robots**: these robots have the same abilities of the partially aware robots; furthermore they are able to auto-diagnose their mobility functionality. In particular, when a fault to the mobility of the robot $R_b^k$ occurs, $R_b^k$ sends the message `MOBILITY-DIAG R_b^k diag` by means of which the behavior mode `diag` of the mobility is conveyed to the Supervisor.

A simple way to model these three different robots categories consists in modeling their self-assessment skills by means of a communicating automaton. The robot Self-Assessment Skills automaton defines the cases when a robot sends a message to the Supervisor as a consequence of a specific detected event; more precisely, we model the ability of a robot to assess a particular condition of the real world by means of the detection of an internal event which triggers a robot message. For example, Fig. 8 shows the self-assessment automaton of a partially aware robot: it is easy to see that the internal events `fault1`, `fault2` and `fault3`, representing the occurrence of a fault to the mobility, do not trigger any message from the robot; the same events trigger a robot message when the robot is fully aware (see Fig. 9). In the rest of the paper we assume that all the robots are partially aware, so they send messages as shown in Fig. 8.
4. The compositional global status model

So far we have defined the basic elements needed to model a multi-robot domain and we have discussed how each of these elements evolves over time as a consequence of the events triggered by robots. However, for monitoring and diagnosis purposes we need an overall transition relation of the entire system.

In order to build such a global transition relation two steps are required. The first step consists in aggregating convenient subsets of basic elements into action-centered sub-models representing all the possible evolutions of each action \( \text{act} \) assigned to a robot \( R_b_k \) (in the following we will refer to the sub-model for action \( \text{act} \) as the action instance transition relation \( \Delta_{\text{act}}(R_b_k) \)).

The second step aims at building the global transition relation from the \( \Delta_{\text{act}} \) of the currently active actions. To this end there are two possible approaches.

A first approach consists in building a monolithic transition relation of the system by composing all the \( \Delta_{\text{act}} \) of the active actions. Such an approach suffers from severe drawbacks. First of all a monolithic transition relation may have a huge dimension, so it would be hard to deal with from a computational point of view. Moreover, in the domains of interest the transition relation changes over time depending on the actions currently assigned to the robots. Therefore every time an action is completed or dispatched, a new global transition relation would have to be computed, and this task would have to be performed on-line in a very efficient way.

The approach proposed in this paper does not construct a monolithic transition relation. The global transition relation \( \Delta \) is just the list of the relations \( \Delta_{\text{act}} \) of the active actions at each time instant. The partitioning of the transition relation makes it easy to update \( \Delta \) simply by adding or removing some of the \( \Delta_{\text{act}} \).

However, we must take into account the fact that relations \( \Delta_{\text{act}} \) are not completely independent because the presence of shared resources introduces interdependencies among concurrent actions. In this case the partitioning of the global transition relation can be performed only by duplicating the models of the resources (as it has been mentioned in Section 3), and by introducing a set of resource constraints in order to maintain consistency.

Figure 10 shows a more detailed view of the Supervisor internal architecture that will be needed for illustrative purpose in the rest of this section and in the following sections.

4.1. Building the relation \( \Delta_{\text{act}} \)

The main aim of the synchronization is to construct a model for a single action which allows the Supervisor to foresee the dynamic evolutions of the action both under the nominal behavior and under a faulty behavior; in the following we refer to this model as an action instance automaton. The action instance automaton results from the synchronized composition of an appropriate set of domain entities. These entities are chosen according to the Action Template which specifies the robot capabilities required to carry out the action as well as the resources involved in the path followed by the robot to perform the action.

The construction of an action instance automaton can then involve the synchronization of an Action Template automaton with Resource automata, Robot Functionality automata, Fixed Sensor automata and a Robot Self-Assessment Skills automaton.

As a consequence of the synchronized composition, the internal events disappear. Thus the composed automaton can be formally defined as:

\[
< Q, \Sigma_{\text{progEvn}}, \Sigma_{\text{faults}}, \Sigma_{\text{obsMsg}}, \Delta_{\text{act}}(R_b_k) >
\]

where:

- \( Q \) is the set of states; in particular, each state in the composed automaton is an assignment of values to the status variables of the involved robot and resources.
A. The composed automaton for the \textit{GoToRoom}(R_{b_{k}}, R_{i}, R_{j}, D_{h}) action. The dotted edges represent non observable transitions since no sensor can detect them (the sensors are in degraded mode).

\[ \Delta_{\text{act}}(R_{b_{k}}) \subseteq (Q \times \Sigma_{\text{progEvn}} \times 2^{\Sigma_{\text{faults}}} \times 2^{\Sigma_{\text{obsMsg}}} \times Q) \]

is the set of transitions (i.e., the transition relation of the composed automaton); each transition has one incoming exogenous event belonging to \( \Sigma_{\text{progEvn}} \) and can consume one or more incoming exogenous events belonging to \( \Sigma_{\text{faults}} \) (i.e., the model allows for multiple faults to occur at the same time instant) and can emit observable events in \( \Sigma_{\text{obsMsg}} \).

In our architecture, the module responsible for building the action instance automaton is the \textit{Model Composition Module} (MCM) depicted in Fig. 10. By exploiting a model library, the MCM is able to single out which models are required for constructing the action instance automaton given an action and a specific robot which must carry the action.

Figure 11 shows the composed automaton for a \textit{GoToRoom}(R_{b_{k}}, R_{i}, R_{j}, D_{h}) action assigned to robot \( R_{b_{k}} \) when \( R_{b_{k}} \) is in the critical area of bed \( Bed_{x} \) at the beginning of the action. This automaton results from the composition of the action template automaton \textit{GoToRoom}() (Fig. 4), of the automaton of the mobility functionality (Fig. 3), of the automata of the resource \( D_{h} \) (Fig. 5) and its associated sensor (Fig. 7), of the automata of the resource \( Bed_{x} \) and its associated sensor\footnote{Models of resource \( Bed_{x} \) and of its associated sensor are similar to those of the door \( D_{h} \).} and of the Robot Self-Assessment Skills automaton of robot \( R_{b_{k}} \) (Fig. 8). In the particular case depicted in Fig. 11 both sensors associated with the door and bed have reduced precision. For sake of readability we identify the automaton states by means of codes, even if each automaton state actually corresponds to an assignment of values to status variables (see Table 1 for the exact correspondence); non observable transitions are depicted with dotted edges.

In the action instance automaton the internal events have disappeared because of composition, thus this automaton does not require further composition with other automata: the automaton represents a model of the specific instantiation of the action and can be used to make predictions on its evolutions. Moreover, the automaton is not deterministic since not all the transitions are associated with observable events and, for several states, multiple transitions are allowed. Let us consider as an example the states 3, 10 and 17 of Fig. 11: a robot which is traversing the transit area of room \( R_{i} \) and is in the OK mode as concerns the mobility functionality, can persist in the OK mode, evolve in the SLOWDOWN mode, or even in the BROKEN mode (represented respectively by the states 3, 10 and 17). It is not possible to directly observe whether a change in the behavioral mode has occurred (and which one).

4.2. Global system status

Intuitively, the \textit{global system status} consists of the status of each robot and resource. Consequently we represent the system status as:

\[ S = (S_{\text{robots}}, S_{\text{resources}}), \]
where \( S_{\text{robots}} \) and \( S_{\text{resources}} \) are further partitioned as follows:

\[
S_{\text{robots}} = (S_{Rb_1}, \ldots, S_{Rb_n}),
\]

\[
S_{\text{resources}} = (S_{Rb_{1, res_1}}, \ldots, S_{Rb_{1, res_1}}, \ldots, S_{Rb_{n, res_n}}).
\]

In the RoboCare case study, each \( S_{Rb_k} \) represents the status of the robot \( Rb_k \) by means of the following status variables:

- \( Rb_k.\text{pos} \): the robot position.
- \( Rb_k.\text{mobility} \): the health status of the mobility functionality.
- \( Rb_k.\text{handling} \): the health status of the handling functionality.
- \( Rb_k.\text{carry} \): if not null indicates the specific object carried by the robot (for example, a meal).
- \( Rb_k.\text{action} \): the action currently assigned to \( Rb_k \).

Variables \( S_{Rb_k, res_j} \) represent the status of resource \( res_j \) w.r.t. robot \( Rb_k \); in fact, as mentioned in Section 3, we associate a copy of a resource automaton with each robot, so that the robot can affect only the internal status of its copy.
4.3. Modeling the time

We model the time as a discrete sequence of instants. We assume that behavioral modes of the robot functionalities can evolve from a time instant to the next one according to specific constraints; these constraints are represented by the Robot Functionality automata. For example, in Fig. 3, the mode SLOWDOWN of the mobility functionality may evolve in one time instant in modes SLOWDOWN or BROKEN, but there is no way to evolve in the mode OK.

The time plays an important role in determining the outcome of an action, i.e., its successful completion or its failure. In particular, since the actions are part of a plan where temporal constraints among them could be defined, the actions have to be carried out within a specific time interval in order to be considered successfully completed.

As pointed out in [3], defining the maximum time duration of the actions in a plan avoids that the Supervisor has to wait for an indefinite amount of time for the completion of an action; in general, if an action is not completed after an appropriate interval, the action can be considered failed (i.e., the action goal is assumed unachievable). In our discussion we model the actions duration with a tolerance interval. To this end we associate each action with two time deadlines: the first one represents the nominal duration of the action, while the second one is the maximal additional amount of time the Supervisor is willing to wait for considering the action successfully completed by the robot. We assume that when an action is dispatched to a robot, the Scheduler provides the Supervisor with both the time deadlines of that action.

The time deadlines of each action are maintained by the Time Manager module (TMM) (see Fig. 10). More precisely, when a new action act is dispatched, the TMM starts two timers, $τ_{\text{delayed}}(act)$ and $τ_{\text{failed}}(act)$. If an action is completed before the expiration of the first timeout, the action outcome is completed on-time; otherwise expiration of timers $τ_{\text{delayed}}(act)$ and $τ_{\text{failed}}(act)$ means that the outcome of action act is completed with delay and failed respectively.

4.4. The system transition relation

The On-line Monitoring module (OMM) needs to estimate the global status $S$ of the system at each time instant. Due to the partial observability, the OMM is in general unable to know exactly the global status and therefore it actually computes a belief state i.e., a set of possible current states.

In order to determine the current belief state $B_{t}$ the OMM requires a representation of the previous one (i.e., $B_{t-1}$) and a global transition relation $Δ$.

As mentioned above, since $Δ$ changes whenever a current action is completed or a new action is scheduled, we maintain $Δ$ partitioned in as many relations as the current actions, or more formally:

$$Δ = (Δ_{act(Rb_{1})}, \ldots, Δ_{act(Rb_{n})}).$$

The partitioning makes it easy to update the global $Δ$ and this advantage is essential for the computational efficiency of OMM. However, the partitioning of $Δ$ does not take into account the dependencies among the current actions, in particular concerning the competition for the use of the resources; thus the predictions made with $Δ$ may contain spurious states where the resources are used inconsistently. To avoid this problem, we have to impose constraints on the use of the resources, such constraints are imposed a posteriori on the belief state computed via $Δ$ in order to filter out all the states where resources are used inconsistently.

For example, the following constraint imposes that only one robot per instant can use the resource $D_{h}$:

$$S_{Rb_{h}, D_{h}}(t) = \text{BUSY} \Rightarrow \forall Rb_{k} \neq Rb_{h}, S_{Rb_{k}, D_{h}}(t) \neq \text{BUSY}. \quad (1)$$

It is easy to see that constraints such as (1) have indirect effects on the number of robots that can be located within the areas of a resource. In the particular case the constraint (1) limits to one the number of robots that can be located within the $D_{h}.CA$ area at the same time.

We denote with $V_{\text{active}}$ the set of variables whose possible evolutions are specified by $Δ$ and with $V_{\text{inactive}}$ the other variables. In particular, a variable is active if it is directly involved in the execution of an action. However, it may happen that not all the status variables of a robot are involved in the current action performed by the robot (e.g., a GoToRoom() action does not involve the handling functionality); or even a robot could be idle (no action is currently assigned to it); in such cases we need some extra knowledge in order to predict the dynamic evolution of the inactive variables.

We provide a simple but effective solution to the problem by assuming the persistence of the values of $V_{\text{inactive}}$ variables. For modeling the persistence of the $V_{\text{inactive}}$ variables we have to distinguish two cases. The
5. On-Line Monitoring Module

We are now in the position to describe the On-Line Monitoring Module (OMM): in particular, we first present the high level algorithm executed by the OMM and then we sketch how this algorithm can be straightforwardly implemented when the belief state and the system transition relation are encoded as Ordered Binary Decision Diagrams ([5]).

5.1. The algorithm

The main loop of the On-Line Monitoring Algorithm is reported in Fig. 12; the algorithm performs three tasks: it maintains a consistent internal representation of the set of possible system states (i.e., the current belief state); detects the completion or the failure of the actions and, finally, activates the DIM.

We assume that the initial belief state $B_0$ and the initial set of scheduled actions $activeActs$ are given as inputs to the algorithm.

At each time instant $t$, the OMM receives messages sent by sensors and robots. Furthermore the OMM receives (from the Time Manager, see Section 4.2) timeout messages regarding the expiration of timers $\tau_{delayed}(act)$ and $\tau_{failed}(act)$.

A prediction $\Pi$ is computed from the previous belief state by applying $\Delta$ (line 1). $\Pi$ consists of a set of tuples $(s', exo, obs, s)$ where $s'$ is a global status belonging to $B_{t-1}$, $s$ is a predicted global status and $exo$ and $obs$ are the incoming exogenous events and emitted observable events associated with the transition from $s'$ to $s$. The $obs$ events represent the predicted incoming messages and consist of messages sent by robots $obs_{robots}$ and by sensors $obs_{sensors}$.

Actual messages coming from sensors ($sensorsMsgs(t)$) and robots ($RobotsMsgs(t)$) are used to confirm or refute the predictions (line 2). In particular, the function $match$ discards the tuples occurring in $\Pi$ where the $obs_{sensors}$ and $obs_{robots}$ parts do not match

![Fig. 12. The on-line monitoring algorithm.](image-url)
with sensorsMsgs(t) and robotsMsgs(t) respectively; consequently the predicted states s occurring in the discarded tuples are not included in B_t. A further filtering on B_t is performed by applying the resource constraints (line 3); such constraints filter out all the states in B_t where the resources are used inconsistently.

The next step is to detect failure and success of actions (lines 4 to 6). First, detection of the actions act which have reached their goal is performed by checking if post(act) holds in each state belonging to B_t (function DetectCompletion()). Then, these actions are partitioned in onTimeActs and delayedActs depending on whether a timeout on t_delayed(act) has previously occurred (line 5). Finally, the set failedActs is determined by taking into account the occurrence at time t of timeouts associated with t_failed(act) where act belongs to the set activeActs doneActs of the actions currently assigned to the robots and which are not completed yet (line 6). If the detection phase reports that at least one action was completed on-time or with delay or failed, the DIM is activated (see next section).

The last phase of the algorithm (lines 9 through 15) concerns the update of the transition relation Δ.

When an action act(Rb_k) is successfully completed (i.e., action outcome equals to on-time or delayed) the global transition Δ is simply updated by removing the relation Δ_action. In case the action act(Rb_k) fails, the relation Δ_action can not be removed from the global Δ. In fact, the robot Rb_k may still be trying to carry out the action, thus Rb_k may use resources and trigger sensors messages.

To handle these situations, we need to introduce the idea of aborting an action to force the conclusion of a failed action; this means that a failed action act continues to be included in the global Δ until it is aborted by the Planner/Scheduler.

The failure of an action act(Rb_k) is an exceptional situation that requires the Planner/Scheduler to take appropriate recovery actions. For sake of simplicity, in this paper we assume that the Planner/Scheduler abort all failed actions. The abort of the failed action act(Rb_k) forces Rb_k to reach a status such that the status of all the resources w.r.t. Rb_k is known to the Supervisor.

We assume that the list abortedActs is given to the Supervisor by the Scheduler (or by a human operator); abortedActs contains all the failed actions which have been aborted and which can now be removed from the global Δ.

Obviously, when a new action act is assigned to a robot Rb_k, Δ_action(Rb_k) is added to Δ.

5.2. Using the OBDDs

Since the synthesis of Δ_action(Rb_k) has to be performed on-line, computational efficiency becomes an issue. We have adopted the Ordered Binary Decision Diagrams formalism ([5]) in order to encode the transition relations of all the domain entities; the composition among these automata is efficiently performed by means of standard OBDD operators ([17]). The OBDD formalism is also used for symbolically (and thus compactly) encoding the belief state B which potentially contains a very large number of alternative estimated states, so that the application of Δ to B (and the filtering based on incoming messages and resource constraints) can again be performed by using standard OBDD operators (see [18]).

Figure 13 shows how the prediction of the current belief state (lines 1–4 of the algorithm in Fig. 12) can be performed by means of OBDDs.

In line 1 the temporary projection O_tempΠ_t is initialized with the belief state at the previous time instant.

The projection is incrementally computed by projecting the evolutions of each active action through a loop over the set of currently active actions (lines 2 and 3).

An abort may be performed by means of a simple recovery plan, or even it may require the intervention of a human operator; it is out of the scope of this paper to investigate issues concerning the Planner/Scheduler strategies.
The calls to function match (Fig. 12, line 2) are converted in two restrict operations respectively at lines 5 and 6 of Fig.13 ($O_{sensorMsgs}$ and $O_{robotMsgs}$ are the OBDD representations of sensors and robots messages respectively). Finally, at line 7, the OBDD $O_{Bi}$ encoding the current belief state is obtained by imposing the system constraints (encoded with OBDD $O_{SystemConstraints}$) to the temporary structure $O_{tempBi}$ by means of the standard OBDD apply operator; namely, the temporary structure $O_{tempBi}$ is intersected with the global constraints, for filtering out all the inconsistent, estimated states.

### 6. Diagnostic Interpretation Module – preliminary concepts

As discussed above, the OMM is able to determine whether an action has been completed on-time, with delay or it has failed. Thus, the outcome of an action is a synthesis of the messages received by the OMM during the execution of that action and it could be exploited by the Planner/Scheduler to take a decision on what to do next. However, an outcome does not specify the reason why something has gone wrong. In order to take more appropriate decisions the Planner/Scheduler should know the causes of an action failure. The task of providing an explanation of a failure is up to the DIM which has to single out which fault(s) of the robot functionalities is(are) responsible for the failure or, alternatively, to identify the occurrence of a competition among robots for accessing a resource (for short we will use the term troublesome interaction to denote this kind of threats). In this section we first present an example of troublesome interaction, then we discuss how these interactions are represented in high level terms. Given these preliminaries, in Section 7, we show how the DIM is able to infer explanations about actions outcomes.

#### 6.1. A motivating example

In the following, we report an example (taken from the RoboCare domain) which shows how an action can fail even if no fault occurs. In fact, a challenging issue of the multi-robots domains we deal with is that an action failure may not only be due to the occurrence of faults in the robot performing the action, but also to troublesome interactions.

Let us consider the set of robots $R = \{Rb_1, Rb_2, Rb_3\}$; each robot in $R$ is initially located within the transit area of the room $R_1$ (i.e., $Rb_{pos} = R_1.TA$), and for each robot the mobility functionality is in the nominal mode OK. Moreover, suppose that at the time $t = 0$, the Scheduler assigns a $GoToRoom(Rb_1, Rb_2, D_{1,2})$ command to each robot $Rb_1$ in $R$ and that for each action $act(Rb_i)$ the timers $\tau_{delayed}(act(Rb_i))$ and $\tau_{failed}(act(Rb_i))$ are set to the time instants $14$ and $16$ respectively. Assuming that a robot needs $10$ time instants for reaching the door $D_{1,2}$ from its current position in $R_1.TA$, the three robots will try to access the door $D_{1,2}$ at the same time instant $t = 10$. In particular, at time instant $t = 10$, the OMM receives three messages from the fixed sensor associated with the door $D_{1,2}$; these messages have the form $\text{ENTER } Rb_i, R_i.D_{1,2}.RA$ (for each $Rb_i \in R$) and represent the fact that the robots have just entered the request area of the door $D_{1,2}$.

Since the resource door can be used only by one robot at a time, the robots have to compete for accessing the resource. We assume that the robots are able to negotiate the resource access and establish in which order they can cross the door. In this way the resource is accessed consistently, since while a robot is accessing the critical area of resource $D_{1,2}$ the others wait in the $D_{1,2}.RA$ area. Let us suppose that the robots negotiate to access the door $D_{1,2}$ in the order $Rb_1, Rb_2, Rb_3$ (for sake of simplicity we assume that the negotiation is instantaneous). Table 2 reports the positions of the robots in $R$ for the time instants $t = 11$ through $t = 16$. At time $t = 13$, the OMM receives the message $\text{LEAVE } Rb_1, R_1.D_{1,2}.RA$; as discussed in Section 5 this message allows the OMM to infer that the robot $Rb_1$ has completed its assigned action, therefore the OMM activates the DIM not only to provide the Planner/Scheduler with a high level interpretation of the action outcome, but also to prune, if possible, the current belief state (see Section 7). At time $t = 14$, the DIM receives the messages $\text{TIMEOUT-Delayed act(Rb_2)}$ and $\text{TIMEOUT-Delayed act(Rb_3)}$; these messages are sent by the TMM, and based on them the OMM infers that the ac-

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
</tbody>
</table>
tions $act(Rb_2)$ and $act(Rb_3)$ cannot be completed on-time but only with some delay.

At time $t = 15$, the OMM receives the message \textsc{leave} $Rb_2$, $R_2$, $D_{1,2}$, $RA$ and infers that the robot $Rb_2$ has carried out its assigned action; however, since the OMM has previously received a timeout message, the outcome of $act(Rb_2)$ is set to \textit{completed with delay}. In this case the DIM is mainly activated in order to provide the Planner/Scheduler with an explanation of the anomalous execution of action $act(Rb_2)$.

Finally, at time $t = 16$ the OMM receives the internal message \textsc{timeout-failed} $act(Rb_3)$; this message indicates that the action $act(Rb_3)$ can no longer be successfully completed, so the OMM considers $act(Rb_3)$ as failed. Also in this case the OMM activates the DIM in order to provide the Planner/Scheduler with an explanation of the failure of $act(Rb_3)$.

In this example a traffic interaction (i.e., two or more robots trying to access the same resource at the same time) arises as a consequence of a poor plan; however the same interaction may arise as the consequence of a fault. For example assume that, under the same initial conditions, the actions are scheduled in the order $act(Rb_1)$, $act(Rb_2)$ and $act(Rb_3)$ at time instants $t = 0$, $t = 2$ and $t = 4$ respectively. In this way (under nominal conditions) the robots will reach the resource $D_{1,2}$ at different time instants so that no troublesome interactions can occur. However, it is possible that during the execution of $act(Rb_1)$, the mobility functionality of the robot $Rb_1$ evolves to \textit{slowdown} as the consequence of a fault, so that robots $Rb_1$ and $Rb_2$ try to cross the door $D_{1,2}$ at the same time, i.e., a troublesome interaction arises among $Rb_1$ and $Rb_2$. Let us suppose that $Rb_2$ suffers a delay as a consequence of the interaction; then, it is possible that when the robot $Rb_2$ tries to access $D_{1,2}$, this resource is used by $Rb_2$, so also $Rb_3$ may suffer some delay. In other words, the initial interaction between $Rb_1$ and $Rb_2$ may have cascade effects also on the robot $Rb_3$.

6.2. Modeling resource competition interactions

From the previous example it is clear that an action failure may occur as a consequence of either a troublesome interaction or of a fault occurring in the robot performing the action, or even as a consequence of both.

For modeling the interactions we have to take into account a lot of challenging issues. First of all, an interaction can be considered as a complex event that involves several robots for a time interval of variable duration. We say that the robots in the set $R$ are competing robots for a resource $res$ if there exist system constraints that limit the utilization of $res$, for example a constraint prevents the simultaneous presence of multiple robots into area $res.CA$.

Furthermore, in most cases, interactions are not instantaneous and their effects can be observed still after their resolution. Moreover such effects may be temporary or persistent. Another challenging characteristic is that an interaction can involve a variable set of robots during its life; as shown in the previous example a traffic interaction may initially involve only the robots $Rb_1$ and $Rb_2$ but, at a subsequent time instant, it may involve also the robot $Rb_3$.

Finally observe that an action $act$ can require more than one resource, therefore the robot $Rb_k$ performing $act$ could be involved in several troublesome interactions.

Dealing with interactions is also problematic since the DIM needs an adequate language to express them. In fact, an interaction can be generally expressed in terms of values assignments to relevant status variables over an adequate time interval; thus for highlighting the occurrence of a troublesome interaction, the DIM should provide the Planner/Scheduler with all the belief states within the time interval under consideration. Considering the example of Section 6.1, the relevant status variables concern the mobility behavioral mode and the positions of the robots $Rb_1$, $Rb_2$ and $Rb_3$, at each time instant from 10 through 16. It is easy to see that it is inappropriate to provide the Planner/Scheduler with explanations in these terms, since the reason of the failure (i.e., the presence of traffic near door $D_{12}$) is not made explicit.

For this reason the DIM should provide the Planner/Scheduler with explanations in high level terms that summarize the robots status variables over a time interval. The high level language for expressing the interactions must clearly be common to the DIM and the Planner/Scheduler. In the following we discuss how it is possible to model interactions by means of such a common language.

The case of omniscient Supervisor For sake of clarity we first discuss how it is possible to define the traffic interaction discussed in the example 6.1, assuming that the environment is completely observable by the Supervisor; moreover we assume that the Supervisor has a complete model of the negotiation strategies used by the robots, so that it knows the order (determined by a priority associated with each robot) in which competing robots access a resource. Note that under these
into account the following aspects:

belief state encodes exactly one system status.

cise status of the system at each time instant, i.e., the very strong assumptions the Supervisor assesses a pre-\n
its priority, the resource Rb

mal definition of traffic that depends on the robot

it captures all the above aspects of the competition

we have defined three de-

The robots suffer from the competition for the

REQUESTED ) in the same time interval.

D

in

out

Rb

k

Rb

1,2 are different. In fact,

k

priority

k

= 1,2 in different ways:

Rb

1 is not affected;

Rb

2 is delayed but it is able to complete successfully its action; Rb

3 can not successfully complete its action.

We would like to define the traffic notion so that

very strong assumptions the Supervisor assesses a precise status of the system at each time instant, i.e., the belief state encodes exactly one system status.

The traffic interaction has to be defined by taking into account the following aspects:

− The robots in R are competing robots since they have requested the resource D1,2 (i.e., S_{Rb_k,D1,2} = REQUESTED) in the same time interval.

− The intervals in which robots in R compete for D1,2 are different. In fact, Rb

1 is a competing robot in the time interval [10,11], while Rb

2 and Rb

3 are competing robots in the interval [10, 13].

− The robots suffer from the competition for the door D1,2 in different ways: Rb

1 is not affected; Rb

2 is delayed but it is able to complete successfully its action; Rb

3 can not successfully complete its action.

In the RoboCare domain we have defined three degrees of traffic (no, moderate and heavy) by means of a traffic predicate. Note that the traffic interaction could be characterized by a different set of qualitative degrees (e.g., the traffic degrees could be only no and yes), but the key idea persists: a degree specifies the impact of the traffic on the outcome of an action.

The traffic predicate related to a resource res has the following form:

res.traffic(degree)_{Rb_k,[l_{in},l_{out}]}.

More precisely, the traffic predicate is instantiated w.r.t. the resource res, the robot Rb

k under consideration, and time interval [l_{in},l_{out}].

Figure 14 shows the definitions of the instantiations of the traffic predicate in terms of the robots and resources status variables where \( \beta(Rb_k,t) \) denotes the number of competing robots that have priority greater than robot Rb

k, at the instant t. In the specific case of the RoboCare domain, the integer constant N is set to 1.

The interval \([l_{in},l_{out}]\) is the time interval in which the robot Rb

k is influenced by the traffic interaction; in general \( l_{in} \) corresponds to the time instant in which the robot Rb

k enters in the request area of the resource res.RA, and \( l_{out} \) corresponds to the time instant in which the robot leaves the critical area res.CA. The predicate instantiation \( \text{res.traffic(no)}_{Rb_k,[l_{in},l_{out}]} \) indicates when the robot Rb

k is not affected by the traffic over the interval \([l_{in},l_{out}]\), that is, when the robot Rb

k has the greatest priority among the other competing robots, or Rb

k is already using the resource. Recall that \( S_{Rb_k,\text{res}}(t') \) represents the status of the resource w.r.t. the robot Rb

k at instant t', thus \( S_{Rb_k,\text{res}}(t') = \text{REQUESTED} \) represents the fact that the robot Rb

k is

\[ \text{res.traffic(heavy)}_{Rb_k,[l_{in},l_{out}]} \equiv \exists t_x \in [l_{in},l_{out}], \beta(Rb_k,t_x) \geq N + 1 \] (4)

\[ \text{res.traffic(moderate)}_{Rb_k,[l_{in},l_{out}]} \equiv (\exists t_x \in [l_{in},l_{out}], \beta(Rb_k,t_x) = 1) \land (\forall t' \in [l_{in},l_{out}], \beta(Rb_k,t') \leq N) \] (3)

\[ \text{res.traffic(no)}_{Rb_k,[l_{in},l_{out}]} \equiv \forall t' \in [l_{in},l_{out}], \beta(Rb_k,t') \neq \text{REQUESTED} \land \text{REQUESTED} \land Rb_k,\text{priority} > Rb_k,\text{priority} \lor S_{Rb_k,\text{res}}(t') = \text{BUSY} \] (2)
in the request area of res and it is competing to access the resource.

The predicate instantiation res.traffic(moderate)\( \text{RES}_{\text{RES},[\text{in},\text{out}]} \) indicates when the robot \( R_b \) has been delayed since exactly one robot has used the resource res before it. Finally, the predicate instantiation res.traffic(heavy)\( \text{RES}_{\text{RES},[\text{in},\text{out}]} \) holds when the robot \( R_b \) has been delayed since at least two robots have used the resource res before \( R_b \).

It is important to notice that the three different instantiations of the traffic predicate are disjoint and complete. That is, given a specific case of traffic interaction and an involved robot \( R_b \), it is possible to precisely state the degree of traffic that affects the robot \( R_b \). In other words, the three traffic degrees induce a partitioning over all the possible traffic situations.

Another example of threat for the progress of the plan is the robot interaction due to an occluded resource, i.e., a robot has mobility broken within the critical area of a resource preventing other robots from gaining access to the resource. Given a resource res, a robot \( R_b \), and a time interval \([\text{in},\text{out}]\), the predicate associated with the occlusion interaction (i.e., res.occ1) can be instantiated with values yes (i.e., the resource res is occluded by a robot in the interval \([\text{in},\text{out}]\)) and no (i.e., the resource is not occluded in the interval \([\text{in},\text{out}]\)). More formally, the predicate instantiation res.occ1(yes) can be defined as in formula (5) of Fig. 15. In particular, the definition states that the resource res is occluded for the robot \( R_b \) in the interval \([\text{in},\text{out}]\) if there exists a time instant \( t_x \) such that \( S_{R_b,\text{RES}}(t_x) = \text{REQUESTED} \) and there exists another robot \( R_b \) whose mobility is \text{BROKEN} and \( S_{R_b,\text{RES}}(t_x) = \text{BUSY} \). Since the occlusion is a persistent condition, for each time instant \( t \geq t_x \), the robot \( R_b \) continues to be located in the res.RA area; the time \( t_{\text{out}} \) corresponds to the instant at which the \( t\text{failed} \text{act}(R_b) \) occurs.

Notice that, in this particular case where the predicate has just two possible values, the predicate instantiation res.occ1(no) can be simply defined as the negation of res.occ1(yes).

As discussed above, the interaction predicates are defined on the status variables at different time instants. The Supervisor maintains a history \( H \) of size \(|H|\) where the last \(|H|\) belief states are collected. The Supervisor evaluates the predicates simply by inspecting the history \( H \). Obviously, the history has to cover the whole duration of each active action, so \(|H|\) can grow up to the maximum duration of the actions.

The case of Supervisor with partial knowledge. It is obvious that the strong assumptions made above are not realistic; the previous discussion has only introduced the main issues concerning the modeling of an interaction, but it cannot provide an actual solution. In fact, the environment is not completely observable and in general the Supervisor does not have a precise model of the robots negotiation strategies. This lack of knowledge introduces a source of non-determinism which has some relevant consequences. An immediate consequence is that the precise definitions of the traffic(degree) instantiations can no longer be used, and must be replaced with approximations, as we will discuss in this section. Other consequences concerning the combined effects of faults and interactions in determining actions outcomes will be discussed in the next section.

There are several aspects that we have to take into account when we shift our attention from an ideal omniscient Supervisor to a Supervisor with partial knowledge. First of all, the system model the Supervisor relies on may not be precise. For example, the robots positions are modeled in a qualitative way by means of areas, and the negotiation strategies are not known. As a consequence the (approximated) definition of the traffic should not mention the robots priorities.

Moreover, since the system is only partially observable, the Supervisor has to handle a belief state instead of a precise system status. Consequently, we can not assume that the Supervisor is always able to assess a specific degree of traffic.

As in the case of omniscient Supervisor, the definition of a predicate instantiation has the general form
res.traffic\(\text{degree}\)\[\text{Rb}_k\]\{\text{in},\text{out}\}, however instead of being precisely defined, a predicate instantiation is approximated by means of a sufficient and a necessary condition. It is worth noticing that the sufficient and necessary conditions slightly modify the corresponding definition of traffic degree as an effect of the actual knowledge on the system. For example, a sufficient condition forces the traffic\(\text{degree}\) definition to hold only for a restricted number of cases that, given the current system observability, unambiguously belong to traffic\(\text{degree}\). Thus, the sufficient condition captures only a subset of the possible cases of traffic\(\text{degree}\) while the necessary condition relaxes the traffic\(\text{degree}\) definition to capture a superset of the possible cases of traffic\(\text{degree}\).

For evaluating sufficient and necessary conditions of a predicate such as traffic, two dimensions have to be taken into consideration:

- the time interval over which a condition may or may not hold,
- the presence of multiple states in the belief state for each time instant in the interval; different states may correspond to different degrees of traffic.

Given the predicate instantiation \(\text{res.traffic}\text{degree}\)\[\text{Rb}_k\]\{\text{in},\text{out}\}, and the associated sufficient condition

\[
\text{suffCond} \Rightarrow \text{res.traffic\text{degree}}\text{Rb}_k\{\text{in},\text{out}\},
\]

we impose that the sufficient condition does not hold iff \(\forall t \in [\text{in},\text{out}], \forall s \in B_t \text{suffCond} \) holds in \(s\), i.e., suffCond must hold in all the states \(s\) encoded in the belief states \(B_t\) in the interval \([\text{in},\text{out}]\). A sufficient condition \(\text{suffCond}(\text{dgr}_j)\) asserts the traffic degree \(\text{dgr}_j\), and consequently discards all the other possible degrees for traffic.

Given the predicate instantiation \(\text{res.traffic}\text{degree}\)\[\text{Rb}_k\]\{\text{in},\text{out}\}, and the associated necessary condition

\[
\text{res.traffic}\text{degree}\text{Rb}_k\{\text{in},\text{out}\} \Rightarrow \text{necCond},
\]

we impose that the necessary condition does not hold iff \(\forall t \in [\text{in},\text{out}], \forall s \in B_t \text{necCond} \) does not hold in \(s\), i.e., necCond does not hold in any state \(s\) encoded in the belief states \(B_t\) in the interval \([\text{in},\text{out}]\). A necessary condition \(\text{necCond}(\text{dgr}_j)\) which does not hold removes the traffic degree \(\text{dgr}_j\) from the set of the possible traffic degrees.

In other words, the problem is to determine the degree of the traffic interaction which has affected an action outcome (the degree no indicates that the action has not been involved in a traffic interaction). Initially all the traffic degrees are possible, however by evaluating the sufficient and the necessary conditions we can reduce this set of alternatives. More precisely, given a predicate instantiation \(\text{res.traffic}(\text{dgr})\text{Rb}_k\{\text{in},\text{out}\}\), when the associated sufficient condition holds, the set of possibilities is reduced to \(\text{dgr}\); on the contrary, when the associated necessary condition does not hold, \(\text{dgr}\) is removed from the set of alternatives. Unfortunately, the sufficient (necessary) condition associated with a predicate instantiation is not always satisfied (unsatisfiable); thus, after the evaluation of the predicates, a subset of interaction degrees is still possible. This is a consequence of the fact that we discard an interaction degree in a safe way: a degree is possible if there exists at least a state \(s \in B_t\) (for any \(t \in [\text{in},\text{out}]\)) which supports it. Discarding interaction degrees in a safe way guarantees the completeness of the diagnostic process, but the number of possible diagnoses may be large.

In the following we exemplify some approximations used in the RoboCare domain. First of all, let us consider the predicate instantiation \(D_h.\text{traffic}(\text{no})\text{Rb}_k\{\text{in},\text{out}\}\) which is associated with the sufficient condition (7) of Fig. 16. The condition (7) means that if there is no robot different from \(\text{Rb}_k\) in the critical or request area of \(D_h\) at each time instant in the interval \([\text{in},\text{out}]\), then there is no traffic near the door \(D_h\) for the robot \(\text{Rb}_k\) in the same time interval. The interval \([\text{in},\text{out}]\) approximates the real influence interval of the traffic interaction, in particular:

- \(\text{in}\) is the time instant at which the robot \(\text{Rb}_k\) enters in \(D_h.\text{RA}\)
– $t_{out}$ depends on the outcome of the action $act$ performed by $Rb_k$; if $act$ is successfully completed $t_{out}$ corresponds to the time instant at which $Rb_k$ leaves $D_h$. RA; if $act$ fails at the instant $t_{fail}, t_{out}$ is set to $t_{fail}$.

Clearly, the sufficient conditions associated with different instantiations of the traffic predicate must be disjoint while this needs not in general be true for the necessary conditions. For example consider the necessary condition (8) associated with the predicate instantiation $D_h. traffic(no)$ (Fig. 16). The condition (8) means that if the robot $Rb_k$ is not affected by traffic near the door $D_h$ in the interval $\{t_{in}, t_{out}\}$, then $Rb_k$ uses and releases $D_h$ in $\{t_{in}, t_{out}\}$. To demonstrate this property it is sufficient to check that there exists a time instant $t_x \in \{t_{in}, t_{out}\}$ such that $S_{Rb_k, D_h}(t_x) = FREE$; in fact, since $S_{Rb_k, D_h}(t_{in}) = REQUESTED$, the status of $D_h$ w.r.t. $Rb_k$ can be free at a time $t_x > t_{in}$ only after $Rb_k$ has used and released $D_h$.

7. Diagnostic Interpretation Module

The DIM has two main tasks:
– provide the Planner/Scheduler with explanations about the actions outcomes in terms of faults in the robots functionalities and/or the high level concepts that summarize the troublesome interactions (e.g., traffic, occlusion),
– prune the current belief state filtering out those states that are not supported by the inferred conclusions about the behavioral modes of some robots functionalities.

Synthesizing Explanations. Let’s consider how the DIM can explain the failure of the action $GoToRoom(Rb_3, R_1, R_2, D_{1,2})$ performed by $Rb_3$ in the example of Section 6.1.

From the Action Template automaton of the $GoToRoom$ action (Fig. 4) it is easy to identify the elements that may affect its outcome: the mobility functionality of the robot and the interactions (traffic, occlusion) that may arise in crossing the door between the source and target rooms.

These dependencies can be captured by the simple causal graph depicted in Fig. 18. Similar (even if more complex) causal graphs can be associated with actions which require more functionalities or resources.

Given the causal graph, we need a theory $\Gamma$ which defines how the combinations of values of the parent nodes influence the values of the child node. In our framework, we assume that the causal knowledge $\Gamma_{acttype}$ (i.e., the causal formulas used for explaining the outcome of an action $act$ of type $acttype$) is expressed by means of a set of logical formulas that have one of the following two forms:

1. $\alpha_1(x_1) \land \ldots \land \alpha_n(x_n) \land \beta_1(y_1) \land \ldots \land \beta_m(y_m) \Rightarrow outcome(z)$
2. $\alpha_1(x_1) \land \ldots \land \alpha_n(x_n) \land \beta_1(y_1) \land \ldots \land \beta_m(y_m) \Rightarrow [outcome(z_1) \lor \ldots \lor outcome(z_l)]$

where: the elements $\{\alpha_i, i \in \{1 \ldots n\}\}$ represent the functionalities required by $act$ (in particular $\alpha_i(x_i)$ means that the functionality $\alpha_i$ is assumed to be in the behavioral mode $x_i$); the set $\{\beta_j, j \in \{1 \ldots m\}\}$ is the set of predicates concerning the troublesome interactions that may influence the outcome of $act$ (in particular, $\beta_j(y_j)$ indicates that the predicate $\beta_j$ is instantiated to the degree $y_j$, e.g., traffic(no) corresponds to the predicate traffic instantiated to the degree no); and finally, $z_1, \ldots, z_l$ are specific values of an action outcome of type $acttype$.

A theory containing only formulas of the first type would be preferred because this type of formulas allows the DIM to infer alternative explanations concerning a specific action outcome through classical abduction. However, due to the partial observability and to unpredictable negotiation strategy adopted by competing robots, it is possible that different action outcomes can be explained by the same assumptions about robot behavioral modes and degrees of troublesome interactions. For this reason we allow also formulas of the second type but prefer explanations derived with formulas of the first type (see below).

Some of the formulas in the theory $\Gamma_{GoToRoom}$ can be derived from the model of the $GoToRoom$ action; in fact, it is easy to see that when the mobility functionality evolves in a permanent faulty mode such as BROKEN, the outcome of the action is necessarily failed. Other formulas can be derived only with additional knowledge that can be learned by means of a set of systematic system simulations. It is important to note that the knowledge resulting from these simulations holds for a class of actions$^8$.

$^6$Assuming that a robot leaves the request area of a resource only after it has used the resource.

$^7$Control mechanisms at design level are required in order to check that different predicate instantiations have disjoint definitions of the corresponding sufficient conditions.

$^8$In particular, $\Gamma_{GoToRoom}$ does not hold only for a specific instantiation of a $GoToRoom$ action which mentions a specific robot or door, but for all the $GoToRoom$ actions.
Figure 17 shows the set of formulas of the causal theory $\Gamma_{\text{GoToRoom}}$. The symbol $*$ is placed where the value of an element (behavioral mode or predicate) does not directly impact the action outcome since the other elements of the antecedent have a stronger influence.

For example, formula (16) states that if the door $D_h$ is occluded, the action fails independently of the actual mode of the mobility and of the presence of traffic.

When the DIM has to explain the outcome of an action, first of all it singles out the time intervals within which the action may be influenced by a troublesome interaction for the access to resources. In our example, $Rb_3$ requires only resource $D_{1,2}$ and the time interval in which $Rb_3$ could be influenced by a troublesome interaction near $D_{1,2}$ is approximated by the interval $[10,16]$ (in fact $Rb_3$ enters into the $D_{1,2}$-RA area at time $t_{in} = 10$ while $t_{out}$ is set to 16 since $\text{act}(Rb_k)$ fails at time 16).

In the second phase, the DIM selects the formulas of the causal theory that explain the failure. In particular, given the $\Gamma_{\text{GoToRoom}}$ theory and the outcome failed, the DIM selects from $\Gamma_{\text{GoToRoom}}$ all the formulas where the outcome failed appears in the consequent (formulas (12), (13), (14), (15) and (16)); in the following we refer to these formulas as the set $\Gamma_{\text{GoToRoom}}(\text{failed})$. Each formula in $\Gamma_{\text{GoToRoom}}(\text{failed})$ is a possible explanation for the failure of $\text{act}(Rb_3)$; however it is possible to discard some of them by evaluating the predicates occurring in their antecedents.

In our particular case, the evaluation of the predicate $D_{1,2}\text{.traffic}$ is able to discard only the instantiation $D_{1,2}\text{.traffic}(\text{no})_{Rb_3[10,16]}$, however this information is not useful since no formula in $\Gamma_{\text{GoToRoom}}(\text{failed})$ requires $\text{traffic(no)}$. Assuming that the sensor associated with the door $D_{1,2}$ is in the nominal mode, the evaluation of the predicate $D_{1,2}\text{.occl}$ disregards the predicate instantiation $D_{1,2}\text{.occl(yes)}_{Rb_3[10,16]}$. Thus, the set of possible explanations for the failure of the $\text{act}(Rb_3)$ consists of formulas (12), (13), (14) and (15).

Selecting Preferred Explanations. In the last phase, the DIM has to order the set of inferred explanations, and provide the Planner/Scheduler with all the inferred explanations ordered according to preference. At present, we have not yet deeply investigated the issues concerning the selection of the preferred explanation(s). A simple solution consists in a criterion of minimality that prefers the explanations where the smallest number of anomalies has to be assumed. Moreover, the explanations can be ranked according to the form of the formulas used to infer them; for example, formula (12) can be considered less preferred than formulas (13), (14) and (15) because of its weak predictive power. Moreover, formula (13) has to assume two anomalies, i.e., the presence of traffic with degree heavy and the $Rb_k$ mobility under SLOWDOWN...
mode, so it is less preferred than formulas (14) and (15) which assume only one anomaly. Finally, the DIM could exploit heuristic knowledge about which events are more likely in the RoboCare domain. For example, the presence of traffic heavy (14) could be considered more likely than the occurrence of a fault to the mobility (15). In this case the explanations for the example could be ordered as [12–15], and provided to the Planner/Scheduler, or to a human supervisor, in this order.

Filtering the Belief State. The DIM is activated also when the outcome of an action is completed on-time. Since it is not relevant to provide an explanation for a normal outcome, the DIM is invoked just because it may filter out some states from the current belief state. In fact, a side effect of the reasoning of the DIM is that some behavioral modes are not supported by the inferred explanations. For example, it easy to see that if a GoToRoom action is successfully completed, the mobility behavioral mode can not be BROKEN, so it is possible to discard all the states in the current belief state where the mobility behavioral mode is assumed BROKEN.

This kind of filtering obviously takes place also when the DIM is invoked with actions that have been completed with delay or that have failed.

8. Experimental results

In this section we report experimental results obtained by running a prototypical implementation of the Supervisor on a test set of 60 cases in the RoboCare domain.

All the tests have been performed using a prototype of the Supervisor implemented in JDK 1.4.2 SE. The OBDDs are made available by means of the JavaBDD package [21] whereas the description of the environment and the initial state of each case are provided to the prototype via XML files. The prototype has been run on a PC Intel Xeon 3.06 GHz, RAM 1 GB, equipped with Windows 2000 OS.

The test cases have been synthesized by means of a software simulator of the RoboCare environment. The main function of the simulator is to provide the Supervisor with the sequence of messages that, in the real cases, would have been sent by robots and sensors; in particular it reproduces the global behavior of the system while the robots are carrying on the given plan even in presence of multiple faults. The simulator allows the designer of a test case to specify:

- the plan the robots have to execute;
- the faults in robots functionalities which will occur during the execution of the plan;
- the environment (rooms, doors, beds and trayracks) where the given plan has to be executed and the system configuration (robots and sensors).

In order to prove the ability of the OMM and DIM to react to anomalous situations, the test cases are synthesized by a human designer and they are characterized by the occurrence of a large number of faults and troublesome interactions.

In the present paper we report experimental data concerning two different environments called Env-A and Env-B. The static aspects of the two environments are reported in Table 3.

It is worth noting that env-B is more complex than env-A concerning all the relevant parameters, and this higher complexity is reflected in the number of variables needed to represent the status of the system. Remember that the status maintains all the status variables of robots, in particular the variables concerning the robots positions and carried objects, and those concerning the behavioral modes of their functionalities, as well as the variables $S_{Rb_k, res_k}$ concerning the status of the resources.

For each environment we have synthesized 30 cases which differ as concerns the plan of actions, the occurrence of faults and of troublesome interactions.

For each case the maximum number of faults is limited to three, that possibly occur simultaneously; moreover some plans are free of troublesome interactions while some other plans cause the occurrence of troublesome interactions due to peaks of requests for the same resource. Table 4 reports the main characteristics of the test cases.

The test set is particularly stressing. First of all the plans have long duration: the average number of ticks is 5 sec.

### Table 3

<table>
<thead>
<tr>
<th># Rooms</th>
<th># Doors</th>
<th># Beds</th>
<th># Robots</th>
<th># Status Multi-valued Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Env-A</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Env-B</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>8</td>
</tr>
</tbody>
</table>

It is worth noting that $S_{Rb_k, res_k}$ is more complex than $S_{Rb_k}$ concerning all the relevant parameters, and this higher complexity is reflected in the number of variables needed to represent the status of the system. Remember that the status maintains all the status variables of robots, in particular the variables concerning the robots positions and carried objects, and those concerning the behavioral modes of their functionalities, as well as the variables $S_{Rb_k, res_k}$ concerning the status of the resources.

For each environment we have synthesized 30 cases which differ as concerns the plan of actions, the occurrence of faults and of troublesome interactions.

For each case the maximum number of faults is limited to three, that possibly occur simultaneously; moreover some plans are free of troublesome interactions while some other plans cause the occurrence of troublesome interactions due to peaks of requests for the same resource. Table 4 reports the main characteristics of the test cases.

The test set is particularly stressing. First of all the plans have long duration: the average number of ticks is 5 sec.

---

$\text{A tick is a unit of time which corresponds to an amount of continuous real time that depends on the sampling rate of the system.}$

We assume that the time needed to complete an action is in the order of minutes, and that the environment can be sampled every few seconds (e.g., 1 tick = 5 sec).
is about 1000 for both environments; in addition, the plans are complex since almost all the robots are active concurrently and the average number of scheduled actions is at least 150 for both environments; this means that the average number of actions executed by a robot is 30.97 and 25.63 respectively in the environments En-A and En-B.

Apart from the occurrence of faults, each case is characterized by a large number of troublesome interactions, and each of them involves a significant number of robots. This kind of situation is quite unusual in normal conditions but we want to test the ability of the Supervisor to cope with plans of poor quality, where many robots are competing for the same resource at the same time. Finally, consider the average number of DIM invocations, in particular those invocations where the DIM provides an explanation for a non-nominal action outcome (i.e., delayed or failed); in the En-A environment the DIM has to provide an explanation for a non-nominal outcome in the 25% of its invocations; whereas in the En-B environment the DIM has to provide an explanation for a non-nominal outcome in the 20.77% of its invocations. These two results show that the consequences of injected faults and troublesome interactions are not instantaneous but last for a while and affect the activities of many robots.

In order to prove the adaptability of our framework, we repeat all the cases with two different degrees of system observability. In particular, we denote with nominalObs and degradedObs the cases when all the sensors of the environment operate respectively in the nominal and degraded mode.

As discussed in Section 5.1, we have implemented the OMM relying on OBDDs because this symbolic representation formalism is suitable for representing a large number of alternative states. However, in general there is no guarantee that even the size of the OBDD does not become unmanageable. Since in our approach the OBDDs are adopted to encode the transition function $\Delta$, the current belief state and the system constraints, the main problems may occur in two particular steps of the algorithm of Fig. 13.

The first critical point corresponds to line 1, when the OMM computes the current belief state by projecting the previous one. However, as reported in Table 5, the size of the $O_{\Pi}$ is well under control because of the partitioning of the global transition $\Delta$ in terms of $\Delta_{act}$ (in fact, the size of $\Delta$ is just the sum of the sizes of all $\Delta_{act}$). Notice that the size of $O_{\Lambda}$ is bigger when the sensors operate in the nominal mode; a larger number of messages are emitted by the sensors with respect to the case of degraded mode and these additional messages require more space to be encoded into an OBDD.

A second potential source of complexity concerns the application of the system constraints to the current belief state $O_{B_i}$ (line 4). In order to maintain the size

<table>
<thead>
<tr>
<th>Table 4</th>
<th>The main characteristics of the test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td># cases</td>
<td>30</td>
</tr>
<tr>
<td># ticks (avg.)</td>
<td>1101.3 ± 157.53</td>
</tr>
<tr>
<td># scheduled actions (avg.)</td>
<td>154.83 ± 28.85</td>
</tr>
<tr>
<td># active robots per instant (avg.)</td>
<td>3.2 ± 0.24</td>
</tr>
<tr>
<td># faults (avg.)</td>
<td>1.5 ± 0.5</td>
</tr>
<tr>
<td># troublesome interactions (avg.)</td>
<td>5.7 ± 1.8</td>
</tr>
<tr>
<td># robots involved in a troublesome interaction (avg.)</td>
<td>3.63 ± 0.45</td>
</tr>
<tr>
<td># DIM invocations (avg.)</td>
<td>205.83 ± 32.9</td>
</tr>
<tr>
<td># DIM invocations for non nominal outcomes (avg.)</td>
<td>51 ± 11.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5</th>
<th>The sizes of the main OBDDs used for encoding environment, actions and constraints, and the average number of states encoded by a belief state</th>
</tr>
</thead>
<tbody>
<tr>
<td>En-A</td>
<td></td>
</tr>
<tr>
<td>nominalObs</td>
<td>degradedObs</td>
</tr>
<tr>
<td># nodes of $O_{\Pi}$ (avg.)</td>
<td>1804.8 ± 47.3</td>
</tr>
<tr>
<td># nodes of $O_{\Pi}$ (avg.)</td>
<td>131.83 ± 9.1</td>
</tr>
<tr>
<td># nodes of $O_{\Pi}$ (avg.)</td>
<td>257.4 ± 0.24</td>
</tr>
<tr>
<td># nodes of $O_{\Pi}$ (avg.)</td>
<td>39.2 ± 8.9</td>
</tr>
<tr>
<td>states encoded by $O_{B_i}$ (avg.)</td>
<td>118.2 ± 10.64</td>
</tr>
</tbody>
</table>
Table 6
The time required by the Supervisor modules

<table>
<thead>
<tr>
<th></th>
<th>Env-A nominalObs</th>
<th>Env-A degradedObs</th>
<th>Env-B nominalObs</th>
<th>Env-B degradedObs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(msec)</td>
<td>(msec)</td>
<td>(msec)</td>
<td>(msec)</td>
</tr>
<tr>
<td>Avg. time OMM per tick</td>
<td>3.13 ± 0.4</td>
<td>3.63 ± 0.43</td>
<td>15.2 ± 2.1</td>
<td>17.4 ± 2.34</td>
</tr>
<tr>
<td>Absolute max time OMM per tick</td>
<td>63</td>
<td>79</td>
<td>109</td>
<td>156</td>
</tr>
<tr>
<td>Avg. time DIM per DIM invocations</td>
<td>0.3 ± 0.16</td>
<td>0.33 ± 0.17</td>
<td>1.26 ± 0.18</td>
<td>1.53 ± 0.23</td>
</tr>
<tr>
<td>Avg. time OMM + DIM per tick</td>
<td>3.43 ± 0.41</td>
<td>3.96 ± 0.5</td>
<td>16.5 ± 2.2</td>
<td>18.93 ± 2.4</td>
</tr>
</tbody>
</table>

It is important to note that the small size of $O_{SystemConstraints}$ small, the encoded constraints are defined only over the competing robots instead of on all the robots of the team.

As expected, different degrees of observability have an impact on the size of the OBDDs: in particular when the observability decreases the size of the OBDDs increases as a consequence of the reduced effects of the messages in filtering out both $O_{B_1}$ and $O_{B_2}$. Nevertheless, the size of both these OBDDs is very small also in the case of degraded observability (as shown in column degradedObs of Table 5).

It is important to note that the small size of $O_{B_1}$ (after imposing system constraints) does not imply that the belief state $B_1$ contains just a few alternative states. Table 5 shows that the average number of alternative states in the environment Env-B is 1901.12 under the hypothesis of nominal observability. More interesting is the case of degraded observability where the number of alternative states has not been precisely determined because a threshold of 100,000 states is in most cases exceeded. However this huge set of states can be very efficiently encoded in an OBDD of small size (in average it consists of 722.9 nodes) which is just a little larger than $O_{B_1}$ in the case of sensors under the nominal mode.

Even when the assumption of nominal observability holds, the number of states encoded into a $O_{B_1}$ is large enough to make impractical for any human or automatic supervisor to inspect all of them for taking appropriate decisions. These experimental results confirm the need of providing explanations in high level terms which synthesize many alternative states.

Concerning time efficiency, the results are quite positive both for OMM and for DIM. For the environment Env-A and in degraded observability the average time taken by OMM is 3.63 msec (see Table 6), and the peak time$^{10}$ is 79 msec. For the environment Env-B the average OMM time and its peak (again in the degraded case) are 17.4 msec and 156 msec respectively. These results show that the OMM can actually work on-line even with a sampling rate well under one second.

For assessing the computational effort required by the DIM, we consider the time needed for inferring the set of all the possible explanations. This time is particularly affected by the time required to evaluate the sufficient and necessary conditions associated with the interactions predicates (e.g., traffic)$^{11}$. Since the average time for inferring explanations requires at most 1.53 msec, we can state that also the DIM is able to operate on-line in the RoboCare domain.

The results reported above show the efficiency of the approach: however we have to test the Supervisor also as concerns its competence, i.e., its ability to find the correct solution. We measure the competence of the Supervisor by comparing the actual reason of a non-nominal action outcome (e.g., completed with delay or failed) with the explanations provided by the DIM. To this end we have partitioned the set of cases into three categories:

- faulty cases: all the cases where there exists at least a fault while troublesome interactions do not occur,
- interactions cases: all the cases where there exists at least a troublesome interaction while faults do not occur,
- faulty-interactions cases: all the cases where both faults and troublesome interactions occur.

A first positive evaluation of the correctness can be derived by the fact that in all the cases where the DIM is invoked for explaining a non-nominal outcome, the actual cause is included among the set of explanations. A more challenging task is to test the competence by taking into consideration only the preferred explanation provided by the DIM. In this way we evaluate the effectiveness of the explanation ranking provided by the DIM.

$^{10}$The maximum time taken by OMM over all 30 cases defined in Env-A.

$^{11}$The evaluation of the sufficient and necessary conditions is efficiently performed by means of standard OBDD operators.
In Table 7 we report the percentage of diagnoses where the preferred explanation selected by the DIM is actually the explanation of a given not-nominal action outcome.

<table>
<thead>
<tr>
<th></th>
<th>Env-A</th>
<th>Env-B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nominalObs</td>
<td>degradedObs</td>
</tr>
<tr>
<td>Faulty cases</td>
<td>97.9 %</td>
<td>97.3 %</td>
</tr>
<tr>
<td>Interactions cases</td>
<td>99.1 %</td>
<td>99.1 %</td>
</tr>
<tr>
<td>Faulty-interactions cases</td>
<td>98.6 %</td>
<td>98.6 %</td>
</tr>
</tbody>
</table>

In Table 7 we report the percentage of diagnoses where the preferred explanation is also the correct explanation. It is easy to see that the DIM is able to select the correct explanation the right one in most cases.

These very positive results have been obtained with a relatively simple preference criterion which prefers explanations involving the minimal number of faults and prefers troublesome interactions to faults. The results in Table 7 show the impact of such a preference criterion: the percentage of correct preferred explanations in the faulty cases is lower than in the two other sets. In fact, when the DIM has to select between an explanation assuming a fault and another one assuming a troublesome interaction, the DIM prefers the second possibility. As expected the competence degree decreases in presence of degraded observability because some of the possible explanations can not be ruled out as the result of the evaluation of the interaction predicates.

9. Discussion and conclusions

In this paper we have presented a model-based approach to the monitoring and diagnosis of multi-robot systems. Both monitoring and diagnosis are hard tasks in the context of multi-robot systems; in fact, the interactions among the system entities are not fixed a priori and can not be easily anticipated; furthermore, the actions performed by the robots may fail as a consequence of such interactions (e.g., concurrent access to critical resources) even if no fault occurs. The monitoring task is the task of interpreting the messages coming from the system (i.e., sensors and robots) in order to keep an internal representation of the system status. In the Supervisor architecture proposed in this paper, the OMM holds a model of the system (global transition relation \( \Delta \)) by means of which it is able to predict all the possible evolutions of the system and the messages the system sends. However, in the multi-robot systems we deal with, the messages sent by sensors and robots depend on the actions the robots are currently performing. Thus every time an action is dispatched or completed, the relation \( \Delta \) has to be efficiently updated. To face this problem we have developed a novel technique which partitions the global \( \Delta \) in as many relations \( \Delta_{\text{act}} \) as the actions assigned to robots so that each relation \( \Delta_{\text{act}} \) models a single action \( \text{act} \). Whenever an action \( \text{act} \) is assigned to a robot, the global \( \Delta \) is updated simply by adding \( \Delta_{\text{act}} \) to \( \Delta \); similarly, when an action \( \text{act} \) is completed \( \Delta_{\text{act}} \) is removed from \( \Delta \). In this way, the OMM is able to automatically adjust the domain model to the current situation.

Unfortunately, since the system is only partially observable, the OMM has to track a set of alternative system states (i.e., a belief state) that may be highly ambiguous (in our experiments the OMM had to deal with belief states containing up to more than 100000 alternative states); for this reason we have adopted the OBDD symbolic representation formalism [5] for efficiently encoding and manipulating the belief state. The benefits and issues in applying OBDDs to diagnosis have been described in [18,19].

Other challenging problems that have been addressed in this paper concern the inference of the diagnosis and the presentation to the Planner/Scheduler. The DIM is the module of the Supervisor which tackles these problems.

The paper shows how the pieces of information encoded within a history of belief states are synthesized by the DIM to infer a diagnosis in high-level terms that can be provided to the Planner/Scheduler.

The synthesis is performed through different steps. The first step (performed by the OMM) is the detection of the outcome of an action which provides a synthesis and a preliminary interpretation of the low level messages sent by robots and sensors during the action execution.

From the outcome of an action the DIM is able to infer a diagnosis (i.e., a set of possible explanations for that outcome), which is represented as a set of alternative assignments to the status variables of the robots. However, a diagnosis expressed in terms of robots status variables is at too low level to be meaningful for the
Planner/Scheduler. For example, we have shown that if an action fails as a consequence of a troublesome interaction this is not made explicit by a diagnosis represented in such low terms. On the other hand, the Planner/Scheduler should know the actual reason of a failure in order to take an adequate recovery action (e.g., re-plan, re-schedule, etc.).

For this reason we have introduced a language for modeling the troublesome interactions. In particular, we have discussed how each troublesome interaction is approximated by means of a predicate associated with a sufficient and a necessary conditions. Moreover, we have described how these conditions are evaluated by inspecting a history of recent belief states. A predicate synthesizes a complex concept which covers several time instants and involves more than one robot. Thus, the predicates are the high-level terms in which the diagnoses have to be expressed.

It is worth noticing that the high-level interpretation performed by evaluating the predicates is an effective way for handling the uncertainty of a belief state (actually, of a history of beliefs). In fact, an high-level explanation is not directly related to a specific state occurring within a belief state. On the contrary, an explanation synthesizes many possible states. Moreover, the strategy adopted for evaluating the predicates guarantees that a potential diagnosis is never discarded.

In this paper the high-level explanations are inferred by assuming that the evolutions of the system depend only on the actions the robots execute and on their health status. The framework could be extended in order to consider that the system may also evolve as a consequence of unforeseeable events. More precisely, we believe that the monitoring is able to cope with unforeseeable events since it is based on the available observations coming from robots and sensors. For example, if the robot \( r_b \) cannot access the resource \( res \) because of an occlusion, the monitoring is able to build a consistent belief state representing the robot \( r_b \) within \( res.RA \) independently of the fact that the occlusion of \( res \) is caused by another robot or by a not-identified object (e.g., a patient or a chair). On the contrary, dealing with unforeseeable events at the diagnostic level may require more modelling work: in particular the abductive inferences should determine in which way a combination of robot faults, troublesome interactions and unexpected events may affect an action outcome. It is worth noticing that synthesizing an appropriate set of abductive rules is not an easy task since different unforeseeable events may affect the actions in different ways.

Another important modelling decision concerns the granularity level adopted for modelling the actions. In principle the whole plan of actions a robot has to execute could be modelled as a single action; however this strategy is not a good solution. First of all, actions lasting for a long time may origin a problem regarding the size of the history since we require that the history has to maintain completely the evolutions of the current actions. Moreover, long actions may delay the detection of anomalies since the outcome is determined after the expiration of the associated deadlines. Therefore we assume that the actions are modelled at a reasonable level of granularity and require that the completion of the actions is observable (i.e., in the action instances relations the transitions leading to a final state must be observable transitions).

An issue that we would like to investigate more deeply in future research concerns the interplay among the Planner/Scheduler and the Supervisor. Providing the Planner/Scheduler with a comprehensible diagnosis is only a part of the problem; in fact, the Planner/Scheduler may require more specific information (for example the robots positions at specific time instants). On the other hand the Supervisor may advice the Planner/Scheduler to dispatch some actions in order to discriminate among a set of alternative explanations.

As a test bed of the methodology, we have referred to the RoboCare domain, where robotic agents provide services for the elderly in an environment partially controlled by fixed sensors. Implementations of the Planner/Scheduler and Monitoring/Diagnosis modules of the RoboCare architecture have been proposed respectively in [7] and [20]. While in [20] the knowledge is represented via rules specific to the RoboCare domain, in the present paper we have adopted a far more powerful technique based on communicating automata; preliminary results concerning the solution proposed in this paper had been presented in [12]. The use of the communicating automata formalism for modeling the basic system entities has also been adopted in [14] for the diagnosis of distributed systems and in [2,11] for the diagnosis of large active systems. However, in such works the global transition relation does not change over time, whereas in our approach we present a strategy for updating the global transition relation according to the actions currently assigned to the robots of the team. Moreover, we provide an additional level of interpretation of the belief states through the DIM.

The proposed framework behaves well when the given plan is carried on by a team of about ten-twenty
robots and the size of the environment does not cause
relevant communication delays, so we can assume the
communication as instantaneous. However, when the
domain under consideration is geographically distrib-
uted (e.g., a Telecommunication Network [14]) or the
number of robots increases (e.g., as in the Traffic Air
Control domain [8]) a centralized approach may not be
the best solution. A natural way for dealing with spa-
tially distributed domains consists in decomposing the
system under consideration into a set of sub-systems.
Each sub-system represents a monitoring and diagno-
sis problem which is solved by a specific software
agent. In general the sub-problems are not completely
independent from one another, so the software agents
need to cooperate in order to achieve a global agree-
ment about the status of the system and the causes of
an action failure.

It is worth noticing that, even if we address a cen-
tralized approach to monitoring and diagnosis, the pro-
posed approach presents also a preliminary decompo-
sition of the system under consideration. In fact the pa-
per describes how to model the evolution of the system
by exploiting a system transition relation which is part-
tioned in as many transitions as the actions currently
performed by the robots. This characteristic opens the
way to multi-agent approaches. Currently we are inves-
tigating a possible multi-agent approach where each
robot of the team is associated with a software agent.
The software agent monitors the actions the robot exe-
cutes and provides explanations whenever the behavior
of an action deviates from the nominal, expected one
(a similar approach is discussed in [8]). In this case,
the belief state of the system is decomposed into a set
of partial belief states: each of them is maintained by a
software agent and represents the set of possible states
in which the monitored robot may be.

An alternative multi-agent strategy decomposes the
system geographically: the domain is ideally decom-
posed in regions, each region represents a sub-system
which is monitored by a software agent (see for exam-
ple [6]).

Acknowledgement

This research is partially supported by MIUR under
project RoboCare.

References

[1] M. Asada, P. Stone, H. Kitano, B. Werger, Y. Kuniyoshi,
A. Drogoul, D. Duhant, M. Veloso, H. Asama and S. Suzuki,
The RoboCup physical agent challenge: Phase-I, Applied Arti-
of large active systems, Artificial Intelligence 110(1) (1999),
135–183.
based diagnosis of planning failures, in: Proc. AAAI’90, 1990,
pp. 318–323.
[5] R. Bryant, Symbolic boolean manipulation with ordered
binary-decision diagrams, ACM Computer Surveys 24 (1992),
293–318.
[6] N. Carver and V. Lesser, Domain monotonicity and the perfor-
ance of local solutions strategies for cdps-based distributed
sensor interpretation and distributed diagnosis, Autonomous
Agents and Multi-Agent Systems Journal 6 (2003), 35–76.
[8] F. de Jonge, N. Roos and J. van den Herik, How to keep plan ex-
ecution healthy, in: Proc. IJCAI, Workshop on Agents in Real-
time and Dynamic Environments, 2005.
[9] A. Drogoul and S. Picault, The microbes project, an experi-
mental approach towards “open collective robotics”, in: Proc.
of the 5th International Symposium on Distributed Autonomous
[10] M. Kalech and G.A. Kaminka, On the design of social diagno-
sis algorithms for multi-agent teams, in: Proc. IJCAI03, 2003,
pp. 370–375.
of discrete-event systems, in: Proc. of the 12th International Work-
sop on Principles of Diagnosis (DX’01), 2001, pp. 95–
102.
diagnosis of multi-agent systems: a model based approach, in:
Proc. of the Sixteenth European Conference on Artificial Intel-
ligence (ECAI’04), 2004, pp. 848–852.
boldly go where no ai system has gone before, Artificial Intelli-
gence 103 (1998), 5–47.
[14] Y. Pencolé and M.O. Cordier, A formal framework for the de-
centralised diagnosis of large scale discrete event systems and
its application to telecommunication networks, Artificial Intel-
ligence 164 (2005), 121–170.
multi-agent diagnosis with spatially distributed knowledge,
[16] N. Roos and C. Witteveen, Diagnosis of plan execution and the
[17] A. Schumann, Y. Pencolé and S. Thiebaux, Diagnosis of
discrete-event systems using binary decision diagrams, in:
Proc. of the Fifteenth International Workshop on Principles of
Diagnosis (DX’04), 2004.

This research is partially supported by MIUR under
project RoboCare.


