On-Line Monitoring and Diagnosis of Multi-Agent Systems: a Model Based Approach

R. Micalizio and **P. Torasso** and **G. Torta** [1]

Abstract. The paper presents an approach for the monitoring and diagnosis of multi-agent systems where mobile robotic agents provide services and partial observability of the environment is achieved via a set of fixed sensors. This kind of systems exhibits complex dynamics where weakly predictable interactions among agents may arise. A model-based approach to on-line monitoring and diagnosis is adopted: while the dynamics of the system components and their relations are modeled via communicating automata, the global system model is factored in a number of subsystems dynamically aggregating a convenient set of component models.
The On-line Monitoring Module (OMM) estimates the possible evolutions of the system by exploiting partial system observability provided by sensors and agents messages and enforces global constraints. 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 robotic agents and/or troublesome interactions among them. As a specific case-study we refer to the RoboCare project [2].

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 [5] have shown that monitoring and diagnosis play a central role since planning depends on the assessment of the status of the system, including failures.
Recent advances in the fields of cognitive robotics and multi-agent systems have paved the way for approaching complex tasks by means of distribution of subtasks among robotic agents and cooperation among them. So far diagnosis in multi-agent environments has received a very limited attention. In general, the approaches to multi-agent diagnosis assume that agents are able to perform local diagnoses (see e.g. [4] and [3]) and to cooperate each other to reach a global diagnosis.
The approach presented in this paper does not assume that agents are able to perform self-diagnosis and to guarantee perfect co-operation. We approach the problem from a different perspective: we monitor the evolution of the system by collecting information coming from fixed sensors located in the environment and by interpreting these pieces of information in order to detect failures and explain them. Additional information on the status of the system is got via messages volunteered by the robotic agents.
The diagnosis of multi-agent systems requires novel techniques. Although basic entities can be modeled by means of communicating automata as already done in the diagnosis of distributed systems ([6]) the diagnosis of multi agent systems is inherently different from the one of distributed systems: the interactions among system entities are not known in advance since they depend on the specific actions currently assigned to the robotic agents. For this reason we introduce a method for dynamically aggregating convenient sets of component models depending on current actions. Moreover some interactions among robots cannot be fully predicted even at run-time, since the robots exhibit some form of autonomy (e.g. for navigation) and therefore the robotic agents may interfere with each other when they need to access the same resources. This phenomenon adds a new dimension to the diagnosis of multi-agents systems: failures are not only due to the occurrence of faults in one or more robotic agents but also to *troublesome interactions*; such undesired interactions may arise either because of competition for resources or because of the presence of faults in one or more agents involved in the interaction.

These characteristics put extra requirements on the capabilities of the diagnostic module which has not only the task of detecting anomalies (i.e. failures) in the behavior of the global system, but also to single out whether the root cause of an action failure is a fault in a robotic agent or a troublesome interaction. It is worth noting that this challenging problem persists even if we assume that a robotic agent is able to self diagnose its faults.

The main ideas in the paper will be exemplified in the context of RoboCare [2], an Italian project involving a number of partners, which aims at studying issues and challenges involved in the design of systems for the care of the elderly that adopt both fixed (mainly sensors) and mobile heterogeneous agents (robots).
The paper is organized as follows. In section 2 we introduce the RoboCare domain and its overall architecture. In section 3 we describe how we model the entities involved in the studied systems, while in sections 4 and 5 we describe in detail the OMM and DIM modules respectively. Finally, in section 6 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 agents, while the overall system is under the control of a Supervisor, responsible both for synthesizing a plan which achieves the goals entered by the user and for controlling the plan execution. A net of sensors located at fixed positions guarantees a partial observability of what is going on in the environment. In [2] the general architecture of the Supervisor is introduced and three main components are identified: the Planner, the Scheduler and the Diagnostic Agent.

[1] Dipartimento di Informatica, Università di Torino, Italy email: {micalizio, torasso, torta}@di.unito.it
[2] This research is partially supported by MIUR under project RoboCare.

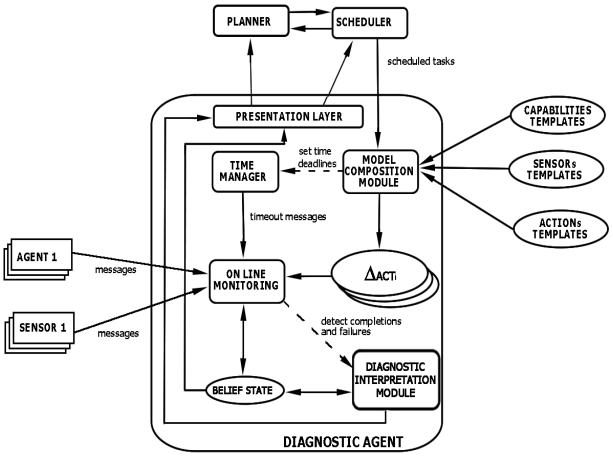

Figure 1. The architecture of the Diagnostic Agent

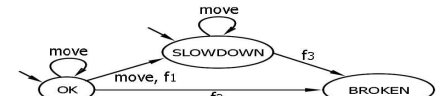

Figure 2.a The automaton of the mobility functionality

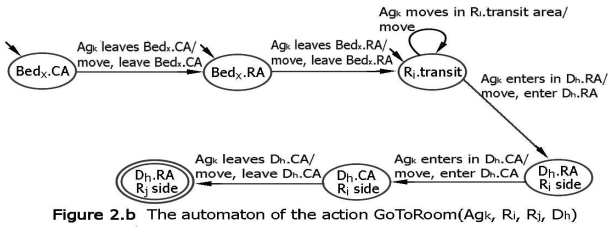

Figure 2.b The automaton of the action GoToRoom(Agk, Ri, Rj, Dh)

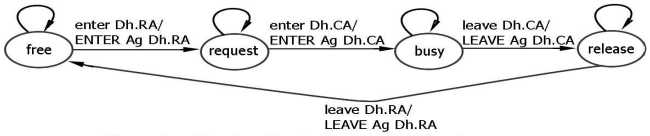

Figure 2.c The functional automaton of the door sensor

Figure 1 shows the architecture of the Diagnostic Agent as described in this paper and its relationships with the other two modules in the Supervisor. We assume that the Planner synthesizes a partial-order plan (POP) where each action is assigned to an agent. 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 the action are satisfied.

The On-line Monitoring Module (OMM) of the Diagnostic Agent is responsible for checking the progress in the execution of the scheduled actions by estimating the set of system states consistent with the observations coming from the net of fixed sensors (and with pieces of information on the health of robotic agents possibly volunteered by robotic agents themselves). Moreover, the OMM detects failures and/or delays in actions execution and in such cases it triggers the DIM which has the task of providing 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 troublesome interactions.

3 MODELING THE DOMAIN

In the following a formal method based on the communicating automata formalism is proposed for modeling the relevant entities of the multi-agent system: robotic agents, fixed sensors and resources.

Environment. We assume that the environment consists in a set of rooms R where relevant objects (such as beds in the Robocare domain) can be located; two adjacent rooms R_i and R_j may be connected by one or more doors D_k.

Since accessing objects and doors is essential for the successful execution of agents actions, we consider objects and doors as *resources*.The positions of resources and agents are modeled in an abstract qualitative way by means of areas within which resources can be placed and agents can move. In particular 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*. The critical and request areas are ideal locations for placing fixed sensors (see below) in order to detect events, in particular agents entering/leaving the area. All the space of a room that is not part of a critical/request area is modeled as a single *transit area TA* used by agents to move from one resource to another. Since no sensor is associated with the transit areas, the Diagnostic Agent cannot observe events occurring within them.

We can express constraints on the concurrent access to a resource directly on *res* or on *res.RA* and *res.CA*; for example, we can limit the number of agents that can occupy an area at the same time.

In the RoboCare scenario, doors and beds are the relevant resources. All the resource constraints are specified on areas; in particular only one agent at a time can access the critical area, while there are no constraints on the request areas nor on the transit area of each room.

Agents. From the Supervisor point of view, the status of a robotic agent is represented not only by a set of variables concerning its position, carried objects, etc. but also by variables concerning health status of its functionalities. In particular, a set of *behavioral modes* is associated to each functionality of the robot; one of them represents the healthy status (*OK* mode) whereas the other behavioral modes refer to faulty states representing degraded or unavailable functionality; for example the robot mobility can be in the *OK*, *SLOWDOWN* or *BROKEN* behavioral modes (see figure 2.a).

A set of available *capabilities* characterizes each behavioral mode of a functionality (e.g. the OK mode of mobility offers the *move* capability). We model an agent's functionality by a communicating automaton where:

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

In the RoboCare domain, the status of a robotic agent is given by its position *pos*, the object it is possibly carrying *holds* and the health status of functionalities *mobility* and *handling*. Figure 2.a shows that the capability *move* is available in behavioral modes *OK* and *SLOWDOWN* but not in *BROKEN*.

Actions. Actions act used by the Planner to synthesize plans are characterized, as usual, by a set of preconditions $pre(act)$ and a set of postconditions $post(act)$. Unlike in classical planning problems, $pre(act)$ and $post(act)$ may put restrictions on the health status of agents functionalities (e.g. a *goto* action requires the *mobility* of the agent not to be *BROKEN*). While at the planning and scheduling level such actions may be considered atomic, for monitoring and diagnostic purposes a more precise description of each action is needed. Since the execution of actions can not be considered as

instantaneous, between the initial state(s) where $pre(act)$ hold and the final state(s) where $post(act)$ hold, there are one or more possible paths touching states where intermediate conditions must hold. Moreover the restrictions on the agents functionalities may be required to hold for *the whole duration* of the action or in some specific intervals, not just in the initial states. In our approach actions are modeled as a communicating automaton such that:

- each state is defined by a list of constant assignments; e.g. in figure 2.b state labels such as $Bed_x.CA$ represent the value assigned to state variable $AGk.pos$

- each transition is labeled with an exogenous input event, this event could be considered as a necessary condition for a transition from state s_i to state s_j. The exogenous events are not directly observable

- each transition is also labeled with one or more internal events that are emitted when the transition is taken. These events may concern agent's capabilities required by an agent to make the transition from s_i to s_j or may describe a state change that is not directly observable (but eventually consumed by the automaton of the fixed sensors - see below). For example, in figure 2.b all the transitions correspond to position changes of the agent and emit a $move$ event intended to be consumed by the automaton of the $mobility$ functionality (if $move$ is an available capability in its current behavioral mode).

Figure 2.b shows the automaton for the action $GoToRoom(AG_k, R_i, R_j, D_h)$ in the RoboCare scenario. In particular the action requires agent AG_k to move from its current position in room R_i to room R_j through the door D_h. Note that the automaton is a *template*, parametric in the arguments of the $GoToRoom()$ action and in the initial position of the agent AG_k; different initial states correspond to different initial positions of AG_k. The intermediate states of the automaton represent, in the sample case, intermediate places where AG_k has to transit in order to reach its final destination $D_h.RA$ in the R_j room.

Fixed Sensors. As we have said, changes in the world status due to the execution of actions are not directly observable by the Supervisor; however, some of them can be detected by fixed sensors which in turn make them available to the Supervisor through messages.

A fixed sensor can have its internal status, and may fail in the same way the functionalities of agents can fail; even if our framework could straightforwardly accommodate the modeling of sensors failures, we take the simplifying assumption that sensors are completely *reliable*. A fixed sensor can thus be modeled as a communicating automaton where each state represents an internal state of the sensor and arcs between two states s_i and s_j receive an event emitted by an action automaton and emit an observable event representing the message sent to the Supervisor.

Figure 2.c shows the automaton for the door sensor in the RoboCare domain, this is a special case where all detected events are made observable to the Supervisor; in general some events may just cause a transition in the sensor internal status.

Fixed sensors must be able to independently detect events triggered by different concurrent agent actions. In order to avoid to define a sensor automaton that considers all possible cases in which two or more events are triggered at the same time, we define each sensor automaton as though no concurrent actions were allowed. We associate a copy of this sensor automaton with each agent, in such way the events triggered by an agent can affect only the internal states of the copy associated to the agent.

4 ON-LINE MONITORING

The On-line Monitoring module (OMM) needs to estimate the global status of the system at each time instant (because of the partial observability, the monitor will actually estimate a *belief state* i.e. a set of possible current states). In order to determine the current belief state $\mathcal{B}_t$ OMM requires a representation of the previous one (i.e. $\mathcal{B}_{t-1}$) and a global transition function Δ

A *global system status* can be described as:

$S = (S_{agents}, S_{sensors})$ [3]

In turn, S_{agents} and $S_{sensors}$ can be further partitioned as follows:

$S_{agents} = (S_{AG1}, \ldots, S_{AGk})$

$S_{sensors} = (S_{AG1,sens1}, \ldots, S_{AG1,sensl}, \ldots, S_{AGk,sens1}, \ldots, S_{AGk,sensl})$

since as already noted we introduce a copy of each sensor per agent. It is quite natural to define Δ starting from the current action of each agent, since the dynamics of the system is determined by the actions currently being performed and by the health status of the agents performing them. For this reason we assume that at any given instant Δ is composed as follows:

$\Delta = (\Delta_{act(AG1)}, \ldots, \Delta_{act(AGn)})$

In order to compute $\Delta_{act(AG1)}$ we compose the action automaton $\mathcal{A}_{act}$ [4] with the automata of the functionalities of AGi and with the AGi copies of the sensors automata involved in act. More specifically, the instantiated action automaton $\mathcal{A}_{act}$ identifies (through the labels of exchanged messages on its arcs) the set of agents functionalities and sensors automata that need to be synchronized with it. Relation $\Delta_{act(AGi)}$ is the transition relation of such composed automaton where all the incoming messages are exogenous and all the emitted messages are observable; thus the automaton does not need any further synchronization with other automata. An important consequence of this choice is that Δ is partitioned by definition on $\Delta_{act(AGi)}$, with huge computational benefits regardless of the actual mechanism used for estimating the next system status.

As an example, suppose that $AG1$ has been assigned a $GoToRoom(AG1, R1, R2, D12)$ action when $AG1$ is located in $Bed1.CA$ area; $\Delta_{act(AG1)}$ results from the composition of the instantiation of the $GoToRoom()$ automaton (figure 2.b), the automaton of the *mobility* functionality of $AG1$ (figure 2.a) and the automata of sensors $sens_{AG1,Bed1}$ and $sens_{AG1,D12}$. In our architecture the module responsible for such computation is the *Model Composition Module* depicted in figure 1.

Since the synthesis of $\Delta_{act(AGi)}$ has to be performed on-line, computational efficiency becomes an issue. We have adopted the Ordered Binary Decision Diagrams formalism ([1]) in order to encode the transition functions of the functionalities automata, the sensors automata and the action automata; the composition among these automata is efficiently performed by means of standard OBDD operators. The OBDD formalism is also used for symbolically (and thus compactly) encoding the belief state $\mathcal{B}$ (which potentially contains a very large number of alternative estimated states), so that the application of Δ to $\mathcal{B}$ can again be performed by using standard OBDD operators (see [7]).

At any given instant, Δ specifies the possible evolutions of the status variables V_{active} involved in the current set of actions. However, it may happen that some agents are idle (no action is currently assigned to them) or some of the functionalities of an agent are not

[3] Also resources should be part of the status, however we assume that their status can be inferred from agents status variables.

[4] It is worth noticing that in order to obtain $\mathcal{A}_{act}$ an instantiation phase is needed since, as we have shown, the action automata are templates (see section 3). Due to space reasons we do not describe this phase.

directly involved in the execution of an action (e.g. a *GoToRoom()* action does not involve the $handling$ functionality); in such cases there are some inactive status variables $V_{inactive}$ not handled by Δ for which we need some extra knowledge in order to predict the dynamic evolution. By assuming the persistence of the previous value of $V_{inactive}$ variables, we provide a simple but effective solution to the problem.

The proposed partitioning of Δ, while essential for computational efficiency, does not take into account the dependencies among the dynamics of the current actions imposed by the constraints on the use of resources. For example, the constraint associated with the critical area of door Dh may state that only one agent at a time can be in that area:

$(AGi.pos = Dh.CA) \Rightarrow (AGj.pos \neq Dh.CA, j \neq i)$

Such constraints must be imposed a posteriori on the belief state computed with Δ.

The following high level algorithm illustrates the OMM process.

loop
  timeouts = timeouts $\cup$ timeoutMsgs(t)
  $\Pi_t = \Delta(\mathcal{B}_{t-1}) = \{(s', exo, obs, s)\}$
  $\mathcal{B}_t = \{s \mid (s', exo, obs, s) \in \Pi_t \wedge$
   $match(obs, sensorsMsgs(t)) \wedge conschk(s, agentMsgs(t))\}$
  $\mathcal{B}_t$ = Prune($\mathcal{B}_t$, resource constraints)
  doneActs = DetectCompletion($\mathcal{B}_t$)
  (ontimeActs, delayedActs) = CheckDelayed(doneActs, timeouts)
  failedActs = CheckFailed(activeActs $\setminus$ doneActs, timeouts)
  if (ontimeActs $\neq \emptyset \vee$ delayedActs $\neq \emptyset \vee$ failedActs $\neq \emptyset$)
    callDIM(ontimeActs, delayedActs, failedActs)
  for each $act \in$ (doneActs $\cup$ failedActs)
    update Δ removing $\Delta_{act(AGi)}$
    update activeActs removing act
  for each new action act assigned to AGi
    update Δ adding $\Delta_{act(AGi)}$
    update activeActs adding act
    set timers $\tau_{delayed}(act)$ and $\tau_{failed}(act)$

At each time instant t, the OMM receives messages form sensors and agents. It also receives (from the Time Manager, see figure 1) timeout messages regarding the expiration of timers $\tau_{delayed}(act)$ and $\tau_{failed}(act)$ that are started when a new action act is assigned to an agent. Expiration of timers $\tau_{delayed}(act)$ and $\tau_{failed}(act)$ means that the execution of act has been delayed and has failed respectively; the actual values of the timeouts are provided by the Scheduler on the basis of a priori knowledge.

A prediction Π is computed from the previous belief state by applying Δ. Π consists in a set of tuples (s', exo, obs, s) where s is a predicted global status, s' is a global status belonging to $\mathcal{B}_{t-1}$ and exo and obs are the incoming exogenous events and emitted observable events associated with the transition from s' to s. Actual messages coming from sensors ($sensorsMsgs(t)$) and agents ($agentsMsgs(t)$) are used to confirm or disconfirm the predictions. In particular, predicted states occurring in Π tuples where the obs part does not match with $sensorsMsgs(t)$ are not included in $\mathcal{B}_t$; in the same way if predicted status s is not in accordance with $agentsMsgs(t)$ it is discarded. For example, if agent AGi volunteers information about its location (AGi.pos=x) and behavioral mode of its mobility (AGi.mob=bm), only predicted states that assign to $AGi.pos$ and $AGi.mob$ values x and bm are kept.

A further filtering on $\mathcal{B}_t$ is performed by applying the resource constraints (all these operations are performed through simple logical operations on the OBDDs encoding the sets of states).

The next step is to detect failure and success of actions execution. First, detection of the actions act that have reached their goal is performed by checking if $post(act)$ holds in $\mathcal{B}_t$ (function $DetectCompletion()$). Then, these actions are partitioned in $ontimeActs$ and $delayedActs$ depending on whether a timeout on $\tau_{delayed}(act)$ has previously occurred. Finally, the set $failedActs$ is determined by taking into account the occurrence in the current time instant of timeouts associated with $\tau_{failed}(act)$. 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 concerns the update of the transition function Δ. In particular, when an action act is completed the corresponding $\Delta_{act(AGi)}$ is removed from Δ; similarly, when a new action act is assigned to agent AGi, $\Delta_{act(AGi)}$ is added to Δ.

5 Diagnostic Interpretation Module

As discussed above, the OMM is able to determine whether an action has been completed on-time, with delay or it has failed. This level of interpretation is not sufficient for the Planner/Scheduler to take a decision on what to do next, since the result of the OMM does not specify the reason why something has gone wrong. The task of providing an explanation of the failure is up to the DIM which has to single out which agent's fault is responsible for the failure or, alternatively, to identify the occurrence of a troublesome interaction among agents.

For performing this task the DIM has at disposal a knowledge base which specifies for each action type $acttype$ the set of entities which influence the outcome of an action act of type $acttype$. In general, the outcome of act is influenced by the health status of the functionalities of the agent performing act as well as by the status of the resources involved in act execution. For example, for a $GoToRoom(AGk, Ri, Rj, Dh)$ action the influencing entities are *AGk.mob*, *Dh.traffic* and *Dh.occl* where *AGk.mob* represents the mobility functionality of *AGk*, while *Dh.traffic* represents the presence of multiple agents trying to cross door *Dh* during the execution of *GoToRoom()* and *Dh.occl* represents the presence of another agent occluding *Dh*.

The knowledge base is partitioned in as many modules as the action types. Each module consists in a set of rules (represented as logical implications) whose consequence is a disjunction of action outcomes; for example the module for *GoToRoom()* contains (among others) the following rules:

$AGk.mob(ok) \wedge Dh.traffic(moderate) \wedge Dh.occl(no) \Rightarrow$
  $outcome(on-time) \vee outcome(delayed)$
$AGk.mob(slowdown) \wedge Dh.traffic(no) \wedge Dh.occl(no) \Rightarrow$
  $outcome(delayed)$
$AGk.mob(slowdown) \wedge Dh.occl(no) \wedge$
  $(Dh.traffic(moderate) \vee Dh.traffic(heavy)) \Rightarrow$
  $outcome(delayed) \vee outcome(failed)$

These rules have a weak predictive power (expressed as a disjunction of predictions) since they contain predicates as *Dh.traffic* that are intended to capture interactions among robotic agents whose effects are not precisely known.

Outcome rules are used abductively taking advantage of the fact that the OMM has provided the DIM with lists of actions subdivided by outcome (see the on-line monitoring algorithm); for example, in case the DIM is invoked at time t and a *GoToRoom()* action *act* occurs in the *delayedActs* parameter, precisely the rules reported above are

activated by the DIM to handle *act*.

The set of abductive explanations that can be found just by knowing that the $GoToRoom()$ action is delayed (according to above rules) are not very strong even if it is possible to rule out that *AGk.mob(BROKEN)*. Stronger conclusions can be reached by evaluating some predicates occurring in the rules; but to do so the DIM needs more information. In particular, the DIM maintains an history H of fixed size r where the last r belief states are collected. Moreover, predicates such as *Dh.traffic* are considered as concepts defined on status variables occurring in the belief states and therefore their evaluation is possible by inspecting the history H. For example the definition of *Dh.traffic(no)* currently adopted is:

$$Dh.traffic(no)_{AGk,t} \equiv \forall t' \in [t-\delta, t], \forall AGi \neq AGk$$
$$(AGi.pos(t') \neq Dh.RA) \wedge (AGi.pos(t') \neq Dh.CA)$$

and it specifies that no agent different from AGk should be in the critical or the request area of Dh and this condition must hold in all time instants of a time interval which ends in the current instant t. It is worth noting that the evaluation of such a predicate is computationally feasible when the presence of agents in Dh critical and request areas is observable for each considered $t' \in [t-\delta, t]$.

Since the DIM can evaluate the predicates *Dh.traffic* and *Dh.occl*, the set of possible explanations is reduced. Let us suppose that in the previous example the evaluations of predicates *Dh.traffic(no)* and *Dh.occl(no)* return *true*: in such a case the only remaining explanation for a delay in the *GoToRoom()* action is *AGk.mob(slowdown)* and this conclusion is forwarded to the Planner/Scheduler (via the Presentation Layer depicted in figure 1). Moreover this conclusion is also used for filtering the current belief state $\mathcal{B}_t$ by disregarding all the global states which involve an assignment to the variable $AGk.mob$ different from *slowdown*.

The possibility of evaluating predicates such as *Dh.traffic* appearing in the premises of rules does not necessarily guarantee that a single explanation is reached. Let us assume that in the example reported above *Dh.traffic(moderate)* and *Dh.occl(no)* evaluate to $true$; in such case we are left with two possible values for $AGk.mob$, namely ok and *slowdown*. In situations like this, the notion of *preferred explanation* can play an important role in ordering the alternative explanations for presentation to the Planner; in particular we prefer to explain the delay on the GoToRoom() action just in terms of the occurrence of *Dh.traffic(moderate)* (with *AGk.mob(ok)*) without further assuming *AGk.mob(slowdown)*.

Finally, it is worth noting that our formalism is able to single-out the root cause of an action failure. For example, let's assume that the execution of $GoToRoom(AG_k, R_i, R_j, D_h)$ has failed. Considering all possible failure explanations the DIM singles out as the preferred one *Dh.occl(yes)*, a predicate defined as follows:

$$Dh.occl(yes)_{AGk,t} \equiv \exists AGi \neq AGk, \exists t' < t$$
$$AGi.mob(t') = broken \wedge AGi.pos(t') = Dh.CA$$

By exploiting this definition, the DIM can extract the root cause: $AGm.mob(t') = broken \wedge AGm.pos(t') = Dh.CA$ (i.e. agent AGm mobility broke in $Dh.CA$ at some previous time t'); so the DIM explains the AGk failure by means of the failure of another agent. The root cause is hence presented to the Planner because it gives more information. Note that this same root cause may be returned as explanation of other actions failures.

6 DISCUSSION AND CONCLUSIONS

In this paper we have presented a model-based approach to the monitoring and diagnosis of multi-agent systems. The first attempts to attack the problem assume that the agents are able to perform self-diagnosis. In the context of *social diagnosis*, [4] advocates a centralized approach assuming a complete observability of the agents behavior and the possibility of querying other agents to get full knowledge of their beliefs. In [3] the system is spatially distributed and naturally divided in subsystems; each agent has a complete knowledge of a subsystem and infers a set of diagnoses only for such subsystem. In order to achieve global consistency, the agents exchange local diagnoses with each other.

Our prospective is quite different. First of all it addresses the problem of interactions arising from concurrent access to critical resources (a problem not faced in previous approaches); moreover, we do not assume that each agent is able to perform self-diagnosis. Therefore, partial observability of the system status is obtained through a net of fixed sensor.

For modeling the system, we adopt the communicating automata formalism, previously proposed for modeling distributed systems where the global transition function is partitioned but does not change over time ([6]). A contribution of the present paper concerns the ability of the OMM to dynamically revise the transition relation Δ to reflect interdependencies among status variables determined by the actions currently assigned to agents; computation of Δ is entirely based on composition of communicating automata each one representing the model of a system entity. We are currently experimenting the adoption of OBDDs for symbolically representing both the belief state and transition function; the benefits and issues in applying this formalism to diagnosis have been described in [7].

In the proposed architecture the DIM is responsible for explaining failures to the Planner/Scheduler, including those caused by troublesome interactions. This is particularly challenging because some interactions among agents are only weakly predictable; in order to manage the problem from a computational point of view, we require a certain level of observability of specific status variables, such as those representing the presence of agents in critical areas.

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 ([2], [8]). While in [8] the knowledge is represented via rules specific for the RoboCare domain, the present paper discusses more general, compositional models and reasoning techniques based on communicating automata.

REFERENCES

[1] R. Bryant, 'Symbolic boolean manipulation with ordered binary-decision diagrams', *ACM Computing Surveys*, **24**, pp. 293–318, (1992).

[2] A. Cesta and F. Pecora, 'Planning and scheduling ingredients for a multi-agent system', in *Proc. PlanSIG02*, Delft, (2002).

[3] N. Roos, A. ten Teije, C. Witteveen, 'A protocol for Multi-Agent Diagnosis with spatially distributed knowledge' in *AAMAS'03*, pp. 655–661, Australia, July 2003.

[4] M. Kalech and G.A. Kaminka, 'On the design of social dignosis algorithms for multi-agent teams', in *Proc. IJCAI03*, pp. 370–375, (2003).

[5] N. Muscettola, P. Nayak, B. Pell, and B. Williams, 'Remote agent: to boldly go where no ai system has gone before', *Artificial Intelligence*, **103**, 5–47, (1998).

[6] Y. Pencolé, M.O. Cordier, and L. Rozé, 'Incremental decentralized diagnosis approach for the supervision of a telecommunication network', in *Proc. DX01*, (2001).

[7] P. Torasso and G. Torta, 'Computing minimum-cardinality diagnoses using obdds', *LNCS*, **2821**, 224–238, (2003).

[8] P. Torasso, G. Torta, and R. Micalizio, 'Monitoring and diagnosing multi-agent systems:the robocare proposal', in *Proc. IASTED Conf on AI and Applications*, pp. 535–541, (2004).