Available online at www.sciencedirect.com
ScienceDirect
Knowledge-Based Systems 20 (2007) 134–142
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

On-line monitoring of plan execution: A distributed approach

Roberto Micalizio *, Pietro Torasso

Università di Torino, Dipartimento di Informatica, Italy

Received 10 October 2006; accepted 16 November 2006
Available online 8 December 2006

Abstract

The paper introduces and formalizes a distributed approach for the model-based monitoring of the execution of a plan, where concurrent actions are carried on by a team of mobile robots in a partially observable environment. Each robot is monitored on-line by an agent that has the task of tracking all the possible evolutions both under nominal and faulty behavior of the robot and to estimate the belief state at each time instant. The strategy for deriving local solutions which are globally consistent is formalized. The distributed monitoring provides on-line feedback to a system supervisor which has to decide whether building a new plan as a consequence of actions failure. The feasibility of the approach and the gain in the performance are shown by comparing experimental results of the proposed approach with a centralized one.

Keywords: Distributed on-line monitoring; Model-based reasoning; Plan execution monitoring; Multi-agent systems

1. Introduction

In the recent years, growing efforts have been spent for providing multi-agent systems with a *closed loop of control feedback* in order to complete the given task despite something has gone wrong (see e.g., [5]). In general these efforts advocate the presence of a system supervisor which synthesizes an initial plan, and possibly adapts on-line such a plan when unexpected events occur. Within the control loop a critical role is played by the activities of *monitoring* and *diagnosis*. In fact, monitoring on-line the progress of the task allows the detection of discrepancies between the expected nominal behavior of the system and the observed one. While the diagnosis is essential for singling out the root causes of the failure of the task carried on by the agent.

Monitoring the activity of software agents as well as robotic ones is a challenging problem, in particular when the actions of the plan are performed by a team of concurrent executors in a complex and dynamic environment where only some events are observable and the executors may fail.

Informally, the task of monitoring consists in tracking the evolutions of the system under consideration (i.e., maintaining a history of system states as accurate as possible) and detecting anomalies whenever they occur. In this paper, we have to monitor a plan with concurrent actions carried on by a team of plan executors. We will assume that plan executors are *robots*, however they are domain dependent, for example in the Air Traffic Control domain (see [9]) plan executors are the airplanes which execute their own flight plan.

As discussed in [1], the successful execution of a plan is threatened by unexpected events which may cause the failure of some actions (the anomalies the monitoring has to detect). In our approach, plan *threats* are faults in robot functionalities or robot competitions which may arise when a number of robots request the same resource simultaneously.

A planner coping with any particular class of domain dependent threats has a choice of either: (1) attempting to prevent threats, or (2) attempting to deal with threats

* Corresponding author.
E-mail addresses: micalizio@di.unito.it (R. Micalizio), torasso@di.unito.it (P. Torasso).

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2006.11.010

individually as they arise [1]. Some approaches to plan monitoring and diagnosis (see e.g., [11]) consider just atomic actions and require that the monitored plan must satisfy a *concurrency requirement* which prevents the occurrence of robots competition for accessing the resources.

We present a system supervisor able to deal with threats when they arise, in this way, given a high-level goal, a planner may synthesize a plan P without the (heavy) request that all the possible threats are prevented. We do not require that the actions are atomic and, as suggested in [1], each of them is associated with a nominal duration: when the actual duration of an action exceeds its nominal time the action is considered *failed*. Moreover, the actions may require some resources that typically can satisfy just a limited number of requests per time instant, thus the execution of the plan must, in general, satisfy a set of *resource constraints*.

While a centralized approach to the monitoring of the execution of plan P is discussed in [6,7], the present paper presents a distributed approach to the problem. In particular, the paper formalizes an approach where the plan is distributed among several plan executors (robots) and each executors is on-line monitored by an agent. In order to establish a control loop, our monitoring framework provides the supervisor not only with the status of the plan but also with the (possibly not nominal) *outcome* of the actions. As we will discuss, the actions outcome is a useful piece of information exploited by the supervisor for taking a decision on whether building a new plan in response to a failure. In general, however, the supervisor would need also to know the root causes of a failure (i.e., a failure explanation). For space limits, in this paper we do not discuss the diagnostic component responsible for fault identification or for singling out specific threats: a possible solution for this problem is reported in [6].

The paper is organized as follows, in Section 2 basic concepts about the distributed plan monitoring are introduced, in Section 3 a formalization of a multi-robot environment is presented, while in Section 4 the distributed approach to the monitoring is discussed. In Section 5, we present some experimental results we have gathered by using the simulated RoboCare environment (see [4]), and compare the centralized approach (described in [7]) vs. the decentralized one.

2. Characterizing the problem

The problem we are interested in concerns the specification of a closed loop of control feedback established through the presence of *monitoring* and *diagnosis* services. In such a way the supervisor has the capabilities for looking after the progress of the plan and for dealing with unexpected threats. In [6] the services of monitoring and diagnosis are performed in a centralized way by the plan supervisor itself, which collects all the available system observations and keeps track of the progress of the actions the robots are performing.

In the present paper, we focus our attention on the on-line monitoring service and we describe how the monitoring of a given plan P can be distributed among a team Ags of software agents. The reason why we propose a distributed approach stems by the observation that a centralized one may result computationally expensive when the number of robots grows. In fact, a centralized approach has to build a global representation of the system status which takes into account all the possible combinations of the robots states. However, given partial observability of the environment, this representation may contain a huge number of alternatives and therefore it could become unmanageable.

We thus propose to decompose the task of monitoring the robot team $\mathcal{T}$ into a set of sub-problems; each sub-problem is assigned to an agent $i \in Ags$ and consists in monitoring just the robot rb_i. The only available observations for agent i are: messages sent by a set of sensors, distributed in the environment, in response to a detected event concerning rb_i, and messages volunteered by rb_i itself about its status (e.g., current position). It is worth noting that in most cases the observations are not sufficient for precisely inferring the status of each robot.

The partitioning described above does not guarantee that the sub-problems are completely independent of one another. In fact, since robot interactions may arise, the actual progress of the action carried on by rb_i depends on the rb_i's health status as well as on the occurrence of robot interactions which involve rb_i. Therefore agent i needs to cooperate with other agents in order to maintain a globally consistent representation of the status of the robot rb_i.

Effective cooperation among agents is reached by adopting two strategies which result to be useful in the context of the distributed problem solving ([3]). First of all, we reduce as far as possible the number of cooperating agents, in particular, each agent $i \in Ags$ determines on-line (i.e., at each time instant) the subset of other agents (denoted as *dependency set*) it has to cooperate with by taking into consideration the actions currently executed by the team of robots. Clearly, since the actions change over time also the relations among the software agents need to change.

The second strategy concerns what sort of data the agents exchange for achieving cooperation. Instead of the rough data that each agent directly receives from the sensors and the robot, the agents exchange partial results which will be subsequently refined by integrating them with the partial results inferred by all the other agents in the same dependency set of agent i.

Whenever an agent detects the failure of an action it informs the supervisor by means of the *outcome* of that action. The outcome of a failed action represents a first kind of data the supervisor can rely on in order to take a decision for overcoming that particular failure. Of course, in case the supervisor revises the original plan P, the new plan needs to be redistributed among the software agents.