Involving the Human User in the Control Architecture of an Autonomous Agent

Roberto Micalizio, Giancarlo Nuzzolo, Enrico Scala, and Pietro Torasso

Dipartimento di Informatica, Università di Torino, Torino, Italy
{micalizio,nuzzolo,scala,torasso}@di.unito.it

Abstract. The paper presents an architecture for an autonomous robotic agent, which carries on a plan in a partially observable environment. A Supervisor module is in charge of assuring the correct execution of the plan, possibly by inferring alternative recovery plans when unexpected contingencies occur. In the present paper we describe a control strategy where a human user is directly involved in the control loop, and plays the role of *advisor* by helping the robotic agent both for reducing ambiguity in the robot's observations, and for selecting the preferred recovery plan.

Keywords: Autonomous Agent, Situation Awareness, Plan Execution, Replanning.

1 Introduction

The adoption of robotic agents is becoming critical in many humans' activities ranging from very challenging scenarios of space exploration and disaster rescue to various kinds of service robotics. However, the effective use of service robots in partially observable environments requires that the robots must be (at least partially) autonomous since the execution of a plan must take into account that actions may deviate from their expected effects as a consequence of unexpected contingencies (e.g., unpredictable changes), or when the actual state of the environment is different from the one originally expected.

Anticipating all the possible contingencies by synthesizing conditional or conformant plans is often unmanageable (or even impossible) as the domain complexity prevents to solve the planning problem efficiently. As a consequence, the robotic agent must be able to recognize unexpected contingencies while it is performing its own actions, and properly react to them (e.g., by changing its plan/goals or by interrupting the plan execution when a global failure is detected).

Recently a number of works have faced the problem of plan execution in partially known environments by proposing different control architectures (e.g. [1,2]).

Other works, such as [3,4], have focused the attention on plan execution monitoring and diagnosis as fundamental activities to detect and explain failures. More precisely, [4] proposes a closed loop of control involving, besides action monitoring and diagnosis, a local re-planner that can be invoked to recover from action failures by synthesizing alternative plans on-the-fly.

M. Bramer (Ed.): IFIP AI 2010, IFIP AICT 331, pp. 13–22, 2010.

While these approaches have been conceived in order to endow the robotic agent with as much autonomy as possible, in several domains (such as rescue, space exploration and UAV control) there is evidence that a more robust behavior can be obtained by having a human user involved in the control loop and allowing the system to operate at different levels of autonomy [5,6].

In the simplest view, the agent can be either fully autonomous, or teleoperated; however there are other intermediate cases where the human can play different roles. In the present paper we consider the case where the human plays the role of *advisor* by solving on-the-fly problems arising during the plan execution that the agent is unable to solve autonomously. Humans, in fact, can help the agent in interpretative activities (e.g., object recognition, situation assessment, etc.), that still are not effectively solved by the current generation of robotic agents. However, in order to support the human intervention during plan execution, the control architecture of the robotic agent must be significantly revised. First of all, the robotic agent must be able to interact with the human user by sending her/him help requests and by understanding human's commands. More important, the robotic agent must guarantee the human with a sufficient level of *situational awareness* [7] by providing relevant pieces of information concerning not only its internal status (e.g., position, health status, etc.), but also that portion of the environment relevant for solving the problem.

2 Control Architecture

The system architecture proposed in this paper, shown in Figure 1, has been inspired by the three tiers architecture proposed by Ghallab et al. [1].

While our Functional Layer (FL), the lowest tier, is similar to the one discussed by Ghallab, the main differences w.r.t. the Ghallab's solution concern the Decision Layer (DL) and the Executive Layer (EL). In the Ghallab's architecture, in fact, the role of the human is limited in submitting goals and retrieving results when they have been achieved; on the contrary, in our proposal the Human User (HU) plays an active role during the plan execution by supporting the DL in its deliberative processes. For this reason we added a Human User Layer (HU-Layer) on the top of the DL as shown in Figure 1. Of course, the existence

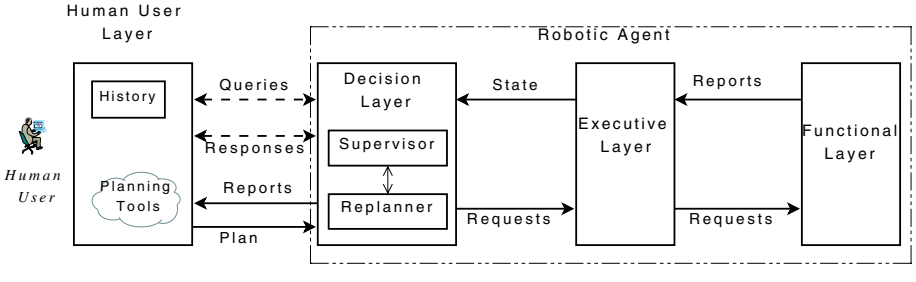

Fig. 1. The extended control architecture

of a HU-Layer forces us to decide when and how the interactions between the HU and the robotic agent RA can occur.

While the proposed architecture supports a Mixed-Initiative approach [8], where both the HU and the RA can start an interaction phase at any time during the execution (see the dashed arrows in Figure 1), for space reasons in the present paper we will focus just on the System Initiative pattern.

In our architecture the HU has the task of providing the RA with an initial plan P (the control architecture does not make any assumption on the way the plan P has been obtained, either manually or via a set of planning tools). The HU-Layer is an intermediate layer between the Human User and the Decision Layer: on one hand it allows the HU to submit the plan P to the RA, on the other hand accepts query and inform messages coming from the DL, and displays these data to the HU so that she/he can answer the requests. The HU-Layer maintains a history of the data received by the DL (i.e. the behavior of the RA and the changes occurred in the environment over a time interval) for increasing the human's situation awareness.

As soon as the DL receives the plan P, DL starts the plan execution phase. Note that the DL includes two modules: a Supervisor and a Replanner; the former implements the control strategy, the latter revises the plan P when unexpected contingencies occur. The Supervisor has a hard task since it has to deal with incomplete knowledge, partial observability and unexpected contingencies. In fact, the plan is synthesized off-line when the actual status of the system is not completely known and assumptions have to be made, but these assumptions could be invalid when the plan is executed. During the execution, the Supervisor has to assess the actual status of the environment via sensing actions. However, the RA has not in general a sufficient amount of information for precisely assessing the actual status of the world (the environment is only partially observable), so it maintains a *belief state* (namely, a set of alternative system states which are consistent with the available observations), rather than a precise state of the environment. The results of sensing actions are used to refine the belief states (i.e. they reduce ambiguity) by pruning off all those states which are inconsistent with them. However, sensing actions are not always so accurate to report a unique interpretation of the environment state and therefore, the result of a sensing action could be still ambiguous. Whenever the actual status of the environment differs from the expected one, the Supervisor invokes the Replanner to repair (when possible) the plan.

The plan. As said above, the user provides the RA with a plan P. We only require that P is a sequence of actions (i.e. a totally ordered plan) where each action is modeled in terms of preconditions and effects. Figure 2 shows an example of such a plan, where the solid edges between actions models causal links; e.g., the edge from action `9` to action `10` is labeled with the service `ONBOARD(R,B1)` means that such a service is an effect of action `9` and a precondition of action `10`; dashed edges between actions model precedence relations.

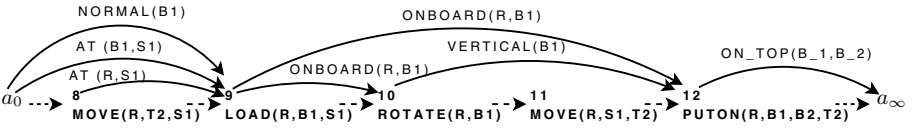

Fig. 2. A segment of the plan P assigned to the robotic agent

3 A Motivating Example

In this section we introduce the scenario we use throughout the paper to exemplify the proposed architecture and control strategy.

We consider an assembly task where a RA has to build a bridge (consisting of two pillars and a beam) by assembling a number of bricks. Each brick has a *length* (short, medium or long), an *orientation* (horizontal or vertical) and a *status* (normal or flawed); only normal bricks can be used to build the bridge. The RA is initially given with a plan P (see Figure 2) to build the two pillars of the bridge one in position `T1` the other in position `T2`. The available bricks are initially located in two repositories `S1` and `S2`; `S1` is the nearest repository to the final location of the bridge and contains a sufficient number of bricks to build the bridge. The RA, positioned in `S1`, can load and unload bricks, move from a location to another and recognize the size of the bricks. Actually, Figure 2 shows just the segment of the plan when the RA loads the brick `B1` in `S1` and puts it (in vertical orientation) on the top of `B2` at the target location `T2`.

Since at the planning phase the actual status of the resources (i.e., the bricks) is not known, the planner assumes that all the bricks are *normal*. Thus, the plan P is executable as long as these assumptions match with the actual status of the environment. The RA performs appropriate sensing actions to check the validity of the assumptions stated in the preconditions of the actions to be executed. For example, the load action requires that the block to be loaded must be *normal*, therefore, before loading `B1` the RA must be sure that `B1` is not damaged. However sensing actions are not always so precise to discriminate between *normal* and *flawed*; and hence the response of the sensing action could be $normal(B1) \vee flawed(B1)$. Thus the RA builds an ambiguous belief state where both values are allowed.

When the RA has to deal with such a kind of ambiguous data, it could adopt a very conservative policy by considering as "faulty" any object that is not surely "healthy"; thus the brick `B1` would be considered as *flawed*. However, this solution may be not acceptable as the RA could waste resources discarding "healthy" objects. The control architecture discussed in this paper mitigates this problem by allowing the RA to ask the HU intervention for reducing the belief state ambiguity; the aim is to exploit the human's interpretative skills that usually are better than the interpretative services of the robot. Whenever an action precondition is not satisfied, the RA must repair its original plan, for example by replacing the damaged brick with a normal one. Also in this case the human user may intervene in order to approve the changes to the original plan proposed by the RA.

```
PlanExecutionSupervisor(plan P, status S, goal G)
01 while (hasActionToPerform(P)) {
02   a = getNextAction(P);
03   S_pre = activateSensors(pre(a));
04   S = updateCurrentState(S_pre,S);
05   executable = checkPreconditions(pre(a), S);
06   if (executable equals unknown)
07     executable = queryHU-checkPreconds(prec(a),S);
08   if (executable) {
09       execute(a);
10       markAsExecuted (a, P);
11       S= updateCurrentState(effects(a),S);
12     if (isRelevant(a)) inform-HU(S, a, succeeded);
13   } else {
14    P = recovery(S,G);
15    if (isEmpty(P))
16      inform-HU(S,a, failed);
17     return failure; } }
18 return success;
```

Fig. 3. The main supervisor function

```
recovery(status S, goal G)
01 plansList = Replanner(S,G);
02 if isEmpty(plansList)
03    return NULL;
04 choice = 0;
05 if (size(plansList) > 1)
06    choice = queryHU-selectPlan(plansList,S,G);
07 return getPlanAt(plansList, choice);
```

Fig. 4. The recovery function

4 The Supervision Strategy

In this section we focus on the control strategy adopted by the Supervisor, and on the relations (i.e., flows of information and requests) existing between the Supervisor and the Human User.

Figure 3 reports the high-level description of the strategy, which receives as input the plan P to be executed, the initial status of the system resources $\mathcal{S}$, and the goal G represented as a set of constraints . The status $\mathcal{S}$ initially represents the *nominal* state of the system resources which has been assumed to synthesize P (e.g., in the scenario described above, $\mathcal{S}$ assumes that all the bricks are *normal*).

Since the plan P has been built under incomplete knowledge, the Supervisor is responsible for checking the preconditions of each action $a \in P$ before executing it. Through `activateSensors` command (line `03`), the Supervisor acquires new pieces of information about the environment. In particular, relying on the preconditions of the action a, the function determines which set of sensing actions must be performed. Note that only those preconditions that have been assumed during the planning phase need to be sensed. Whereas, services provided by other actions previously executed can be checked just by inspecting the current status $\mathcal{S}$. As said above, the result of a sensing action may be ambiguous, therefore the response of `activateSensors` command is in general a set $\mathcal{S}_{pre}$ of alternative states.

The new pieces of information acquired by sensing actions are used to update the current system status $\mathcal{S}$ (line `04`). Of course, since $\mathcal{S}_{pre}$ may be ambiguous, $\mathcal{S}$ may be ambiguous in its turn. Therefore, when the Supervisor checks the preconditions of action a in the current status $\mathcal{S}$ (line `05`), the response could be *unknown*: $\mathcal{S}$ includes both states where the preconditions of a hold and states where they do not.

The Supervisor tries to disambiguate its belief state $\mathcal{S}$ by asking the HU (line `07`) to interpret the collected data. To this end, the Supervisor provides the HU with the preconditions of a still to be checked, and the current (ambiguous)

state $\mathcal{S}$; thereby the HU can gain the situation awareness necessary to accomplish her/his task. In general, however, we cannot assume that the HU is always able to disambiguate $\mathcal{S}$, and hence also the HU may reply *unknown* to the Supervisor (in this case the state $\mathcal{S}$ remains ambiguous).

Only when the action preconditions are satisfied for sure (i.e., when the preconditions are satisfied in each state $s \in \mathcal{S}$), the action is executed (line `09`), and the status $\mathcal{S}$ is updated with the action effects (line `11`).

In principle, the Supervisor should notify the HU with plan progresses by invoking the `inform-HU` primitive after each action execution. However, in order not to overload the HU with action feedbacks, this happens only when the action a is "relevant" w.r.t. some user-defined criteria. A simple but effective strategy we have adopted in our tests consists in considering as relevant an action a iff at least one of its effects is part of the final goal.

When the action preconditions are not satisfied (or it is impossible to precisely evaluate them), the RECOVERY module is activated (line `14`). The recovery module (whose main steps are showed in Figure 4) is based on a conformant re-planner, which produces a list of alternative plans reaching the goal G given the current state $\mathcal{S}$.

Of course, when *plansList* is empty no solution exists and the recovery terminates with a failure. In such a case the detected failure cannot be autonomously recovered by the RA, and a human intervention is required. Conversely, when *plansList* contains a number of plans, the Supervisor asks (line `06`, Figure 4) to the HU the preferred plan, the selected plan is used by the Supervisor (line `14`, Figure 3) as the new plan P the RA will execute.

Re-planner. The conformant re-planner invoked by the Supervisor during its activity is the same we have discussed in [9]; such a planner has some similarities with the Universal Planner by Veloso et al. [10], since it goes forward from the initial state $\mathcal{S}$ to the goal G, and exploits the symbolic formalism of Ordered Binary Decision Diagrams (OBDDs) to encode action models and $\mathcal{S}$. The main advantage of our implementation based on OBDDs is that we can avoid backtracking by keeping in just one OBDD (in a compact way) all the possible partial plans built so far: at each step the planning algorithm extends every partial plan so that the search goes on in parallel along different candidate solutions. In this way the extension of the plan of one action is realized in just one shot by exploiting the standard OBDDs operators, and there is no need to iterate over the space of the current partial solutions.

As a consequence, when the goal G is achievable, the re-planner finds all the plans achieving the goal G with the minimal number k of actions (i.e., all these plans are optimal in terms of the number of used actions). The resulting plans are subsequently returned to the RECOVERY module, which diplays them to the HU.

For the specific purpose of our architecture, we have slightly modified the planner in [9] in order to look for alternative solutions in those cases where just one optimal solution exists. However, these further solutions cannot employ more than $k+\delta$ actions to reach the goal, where δ represents the maximum cost (in terms of number of actions) the HU is willing to accept for a non optimal

solution. The idea is to propose to the user a number of solutions (even not necessarily optimal) that helps her/him in taking a decision. In fact, proposing just one plan, even if optimal, may be too restrictive: the HU could consider the choice of that plan just as a decision made by the Supervisor. Conversely, by proposing also not optimal solutions the HU can envisage different scenarios and take a decision based on different biases.

Note that, since the re-planner is involved in an on-line interaction with the HU, its response time should be compatible with the typical times of Human Computer interaction. The preliminary results we have collected[1] suggest that our planning techniques, implemented in JAVA, can meet such a requirement. In fact, when a solution shorter than 10 actions exists, the re-planner takes less than 1 sec. to find it. In more challenging situations, the re-planner finds conformant plans including up to 20 actions in about 30 sec.; note, however, that it is often undesirable having recovery plans longer than 10 actions: when a recovery plan becomes too long the human intervention may represent a more efficient solution.

Communication primitives. Since the HU is involved in the control of the plan execution, the supervision strategy encompasses some specific communicative acts which assure the HU with the right level of situation awareness. The HU, in fact, can effectively intervene only when she/he knows the relevant aspects of the system status (RA and environment). In order to meet these objectives, the supervision strategy exploits two communication primitives *inform-HU* and *queryHU-...*[2] The purpose of *inform-HU* is to provide, step by step, the HU with some information to let him aware of the current progress of the plan. The HU-Layer incrementally collects in a history the pieces of data received via *inform-HU* messages. The HU can therefore gain his/her situation awareness relying upon a recent history of system states . Another communicative act is the help request, modeled by the *queryHU-* primitive, issued by the RA when failures or ambiguities arise; it consists in sending the current state $\mathcal{S}$ and the key data concerning the impasse (e.g. alternative plans, or cues about ambiguities arising during the check of the preconditions), and in waiting for the HU's answer.

Note that the *queryHU-* primitive supports both "management by consent" and "management by exception": in the former the RA waits indefinitely a human's reply while in the latter the RA waits for a pre-defined time interval after which it decides on the safest possible solution (e.g., when the agent is unsure about the status of a brick, it assumes that the brick is unusable).

5 Running Example

Let us consider again the segment of plan shown in figure 2, and let us assume that the next action to perform is the load action `9`. The action preconditions

[1] The experiments have run on a laptop Intel Core2 2.16 GHz, 2 GB RAM.

[2] These two primitives resemble the "inform" and "query" primitives defined in the FIPA-ACL standard (www.fipa.org).

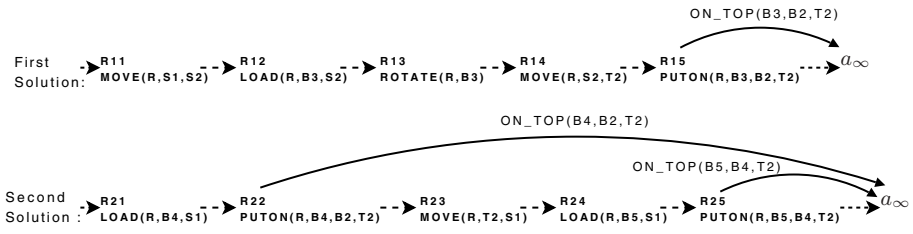

Fig. 5. The two alternative plans to resolve the impasse *flawed(B1)*

require that the brick `B1` must be *normal*, moreover both `B1` and the RA must be within the repository `S1`. It is worth noting that the RA is in `S1` for sure as a consequence of the preceding move action `8` (this atom can be checked by considering the current belief state); whereas both the position of the brick and its *normal* status must be checked via some sensing actions, as these two atoms are assumptions made in the initial state.

Let us assume that, as a result of its sensing actions, the RA Supervisor discovers that the brick `B1` is actually located in `S1`, but it is not normal. The Re-planner module finds two possible solutions, (the recovery plans are reported in Figure 5): the first plan, involving the minimum number of actions, consists in finding another brick, `B3`, whose dimension is exactly the same as the flawed brick `B1`; however `B3` is located in repository `S2`, which is far away from the target positions `T1` and `T2`. Therefore, the cost of the move action `R11` from `S1` to `S2` may be relevant. However the planner provides a second solution consisting in assembling a number of small bricks for obtaining one of the desired size. Since the flawed brick `B1` is *medium* sized, it can be substituted by composing two *small* bricks `B4` and `B5`, both located in the repository `S1`.

These two solutions are presented to the HU, who can determine the most suitable possibly by considering not only the data provided by the RA, but also the knowledge she/he has about the whole environment where the RA operates, which in general is not completely captured by the RA.

6 Implementing the Supervision Strategy via PLEXIL

To implement the Supervisor's strategy we have used the PLEXIL declarative language developed by the NASA [11] (see http://plexil.wiki.sourceforge.net). PLEXIL is a declarative language which allows to hierarchically decompose complex tasks into sub-tasks. A PLEXIL model is therefore a tree, where a node may be activated by other nodes previously executed or by the occurrence of exogenous events. PLEXIL helps the programmer in the definition of the complex task to solve as it allows a clear distinction between the task decomposition and the actual implementation of the required services.

For example, the tree in Figure 6 summarizes the supervision strategy previously discussed: while the intermediate nodes define the task decomposition,

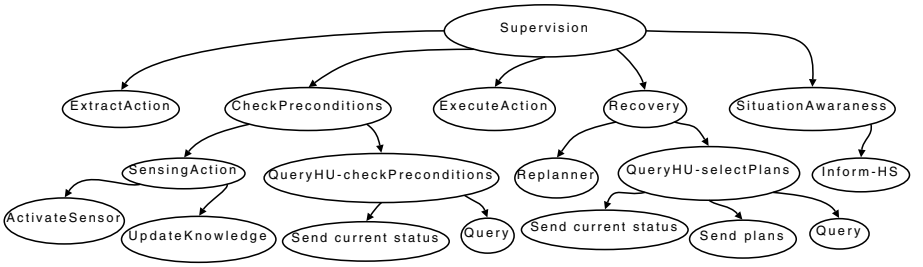

Fig. 6. A graphical representation of the PLEXIL program implementing the Supervisor algorithm

the leaves represent calls to services (such as the Replanner) or to external subsystems (such as the HU-Layer). For the sake of readability the figure reports just the task decomposition, and does not show the causal dependencies existing among the nodes. At the upper level the Supervision task is decomposed into the following nodes: *ExtractAction*, *CheckPreconditions*, *ExecuteAction*, *Recovery* and *SituationAwareness*.

The *CheckPreconditions* node is activated just after the next action to be performed has been extracted and consists of two sub-activities. The former gathers new pieces of information from the sensors and updates the current belief state. The latter, activated only when the previous sensing action returns ambiguous data, is the request for help toward the Human User. The interaction with the human is quite complex, in fact the node *QueryHU-checkPreconditions* is in turn subdivided into two nodes: *Send current status* which aims at providing the human with the pieces of knowledge relevant to describe the impasse, and *Query* which formulates a question to the human, captures the human's answer, and updates the current belief. Note that, the *Query* node takes also care of the management by consent and by exception we have previously described (i.e., a query node is not necessarily blocking).

The *CheckPrecondition* node activates the *ExecuteAction* node iff the precondition of the next action are satisfied in the belief state resulting after the sensing actions. The *ExecuteAction* node delegates the execution of the action to the EL that, in turns activates the services offered by the FL.

The *Recovery* sub-tree represents all the set of operations the RA performs when the action preconditions are not satisfied for sure. The *Recovery* node invokes first the Replanner algorithm previously discussed, and then the *QueryHU-selectPlan* node representing the help request toward the HU; in particular, such a node is subdivided into three other primitive nodes: the actions for sending the current belief state, the set of possible plans (these pieces of information contextualize the problem) and the action representing the real query waiting for the human response.

Finally, the *SituationAwareness* node is activated whenever relevant data have been collected and the user has to be informed about them.

7 Conclusions

The paper has addressed a control architecture for an autonomous robotic agent, which represents a first step towards the development of a more robust robotic agent. In fact, albeit the robotic agent is endowed with some form of autonomous behavior (which enables the agent to face some unexpected contingencies), the agent can request the help of the human user whenever the human, playing the role of *advisor*, can solve the impasses more effectively.

In this first attempt, we have shown a simple form of cooperation between the human user and the agent. On one hand, the computational power of the agent is used to present to the human a range of possible (conformant) solutions for an impasse. On the other hand, the human has the ability to assess them and to select the best one also by exploiting her personal experience without having to compute by himself/herself the conformant plans.

So far the interaction primitives supporting the human's situation awareness are system initiative based; however, we are currently extending them in order to develop more complex forms of human-agent cooperation including mixed initiative interactions (i.e., where the human could intervene without an explicit help request from the agent).

References

1. Alami, R., Chatila, R., Fleury, S., Ghallab, M., Ingrand, F.: An architecture for autonomy. International Journal of Robotics Research 17(4), 315–337 (1998)
2. Galindo, C., Fernndez-Madrigal, J., Gonzlez, J., Saffiotti, A.: Robot task planning using semantic maps. Robotics and Autonomous Systems 56, 955–966 (2008)
3. de Jonge, F., Roos, N., Witteveen, C.: Primary and secondary diagnosis of multi-agent plan execution. Journal of Autonomous Agent and MAS 18, 267–294 (2009)
4. Micalizio, R., Torasso, P.: Monitoring the execution of a multi-agent plan: Dealing with partial observability. In: Proc. of ECAI 2008, pp. 408–412 (2008)
5. Sellner, B., Simmons, R., Singh, S.: User modeling for principled sliding autonomy in human-robot teams. Multi-Robot Systems. From Swarms to Intelligent Automata 3, 197–208 (2005)
6. Fong, T., Thorpe, C., Baur, C.: Collaboration, dialogue, and human-robot interaction. In: Proc. 10th International Symposium on Robotics Research (2002)
7. Endsley, M.: Towards a theory of situation awareness in dynamic systems. Human Factors 37(1), 32–64
8. Bresina, J.L., Morris, P.H.: Mixed-initiative planning in space mission operations. AI Magazine 28(2), 75–88 (2007)
9. Micalizio, R., Torasso, P.: Recovery from plan failures in partially observable environments. In: Research and Development in Intelligent Systems XXVII, pp. 321–334 (2007)
10. Jensen, R.M., Veloso, M.M.: Obdd-based universal planning for synchronized agents in non-deterministic domains. Journal of Artificial Intelligence Research 13, 189–226 (2000)
11. Verma, V., Estlin, T., Jonsson, A., Pasareanu, C., Simmons, R., Tso, K.: Plan execution interchange language (plexil) for executable plan and command sequence. In: i-SAIRAS 2005 (2005)