

Fitting Autonomy and Mobile Agents

Walter J. Vieira

Dept. Telec. and Computer Science
Inst. Sup. de Engenharia de Lisboa
Rua Conselheiro Emídio Navarro
1949-014 Lisboa
Portugal
wv@deetc.isel.ipl.pt

L. M. Camarinha-Matos & L. Octavio Castolo

New University of Lisbon
Faculty of Sciences and Technology
Quinta da Torre
2825-114 Monte Caparica
Portugal
{cam,lgc}@uninova.pt

Abstract - This paper describes the concept of fitting autonomy for remote operation systems, which means that the autonomy of the remote place is defined in a case-by-case basis, taking into account the requirements of the applications and the characteristics of the communication channels. Thus high levels of flexibility and independence on the characteristics of the communication channels are achieved. An architecture for mobile agents equipped with execution supervision is adopted as a solution to implement this approach. An application of these concepts in the context of the remote manipulation of a robot through the Internet is described.

I. INTRODUCTION

Remote operation has been a hot research topic in last decades due to its large range of application domains, such as operation of unmanned spacecrafts and other spatial artifacts [12], operation of machinery in hazardous or man-inaccessible environments, remote operation of telescopes, remote operation of home appliances [2][3], etc. More recently, a tendency to use the Internet as the communication infrastructure for remote operation has emerged in the more traditional fields of telerobotics [1], but also in new application domains such as electronic commerce, information gathering, elderly care [8], etc. This tendency is now spreading over the wireless communication networks.

In cases where a large number of clients of the services provided by a remote place exist, it is difficult to anticipate all the services that all the users will ever need. This means that the achievement of the maximum levels of flexibility requires the implementation of low-level commands in the remote places. Unfortunately the characteristics of the communication channels, such as time-delays (resulting from the propagation time or bandwidth limitations) and intermittency may impose severe constraints on the implementation of this strategy:

- Time-delays contribute to the increase of the completion times of the tasks.
- Time-delays may originate operator distraction.
- Large time-delays may cause instability in the closed loop control system.
- Intermittency or time-delays may difficult the adoption of recovery strategies if errors occur. This is due to the

fact that intelligent recovery strategies depend not only on the single low-level command being executed but also on the overall task being performed.

- Intermittency or time-delays may degrade the response to critical events in the remote place.

In general, the proposed solutions to reduce the dependency on the characteristics of the communication channels have been based on the increase of the autonomy of the remote places. This means that instead of implementing the commands corresponding to the basic functionality of the resources located in the remote places, high-level commands are implemented, thus reducing the needs of a stable and time-delay free connection between the local and the remote places. However, this solution may represent a drastic reduction of the flexibility of the system, since now it is necessary to be able to anticipate all high level commands that will ever be required by all the users. Furthermore, this solution does not take into account that in many situations a lower level of commands may be necessary in order to allow the achievement of more precise control over the task being performed.

In previous work [7], [8], [11] a solution which allows the autonomy of the remote places to be defined in a case-by-case basis, taking into account the needs of the applications and the characteristics of the communication channels was first introduced by the authors. In this paper previous results are further extended, and a detailed description of the implemented architecture and a set of demonstration examples is presented. The proposed solution, called fitting autonomy, is based on mobile agents, which are sent from one local place to the remote place in order to implement there the desired level of autonomy. Since higher levels of autonomy may be required there is the need for execution of supervision mechanisms, which allows the agents to monitor the execution of the tasks being executed, detect deviations from normal execution and react to errors and other events.

The rest of this paper is organized as follows: in section 2 a fast walk through the traditional approaches for remote operation is done; section 3 is reserved for the presentation of the concept of fitting autonomy and how it

can be achieved using mobile agents; in section 4 the results of experiments carried out to evaluate the advantages of the proposed solution in commanding an assembly robot through the Internet are described; finally, in section 5 some conclusions are presented.

II. TRADITIONAL APPROACHES TO REMOTE OPERATION

A. Levels of autonomy

The area of telerobotics is one where the biggest research efforts on remote operation have been done [4]. In telerobotics an operator located in a local site commands a remote executor located in a remote site. The level of autonomy of the remote executors can be classified according to one of the following classes [5]:

- *No autonomy.* This is the case used for manual operation of the remote executor. The intelligence of the remote executor is limited to the execution of the individual low-level orders sent by the operator who is also responsible for the adoption of the recovery strategies when errors occur and to react to events.
- *Full autonomy.* In this case the remote executor has full autonomy to execute high-level orders sent from the local place, and therefore includes the capacity to recover from errors and to react to events.
- *Supervised autonomy.* In this case the remote executor has the level of intelligence necessary to guarantee the execution of intermediate level orders. In general, the operator takes control when errors occur.

The second level of autonomy is difficult to achieve in practice. On the other hand, the first level is appropriate only when the characteristics of the communication channels have no relevance and low-level orders are necessary (such as in precision approaching maneuvers). Most of the telerobotics applications use supervised autonomy. However, it is worth to mention that the evolution from no autonomy to full autonomy is continuous rather than discrete, as it is illustrated in Fig. 1. As a consequence many forms of supervised autonomy can exist.

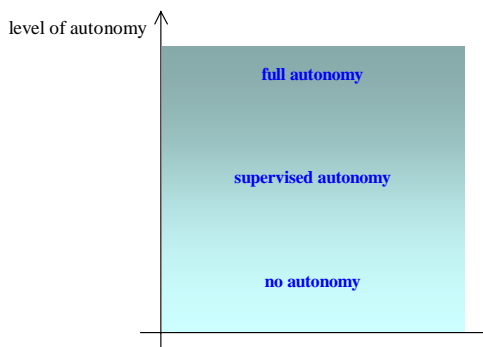


Fig. 1 – Evolution of the levels of autonomy

B. Traditional strategies

Several works in the telerobotics field have been concentrated on attempts to attenuate the dependencies on the time-delays imposed by the communication channels and considering the remote manipulation of a robot arm. Among the many proposed strategies the following can be found:

- *Move and wait.* This strategy is used when the remote executor has little autonomy. For each command sent to the remote place, the operator waits until the corresponding feedback information is received.
- *“Go to waypoint”.* In this strategy the remote executor is endowed with the ability to evolve by its own means to a given reference point. This strategy is commonly associated with supervised autonomy.
- *Predictive displays.* This strategy uses a simulator of the remote executor located in the local place whose execution is advanced in time in respect to the execution of the remote place by an amount of time equal to the time-delay. The image of the position of the remote execution generated by the simulator is superimposed on the image from the current position of the remote executor obtained by telemetry. The operator observes the simulator, which gives immediate response to his stimulus and simultaneously commands both the simulator and the remote executor (Fig. 2). According to [6] results of experiments conducted with this strategy confirm that the performance of the operators is raised and the effect of the time-delays on the completion times of the tasks is highly reduced.

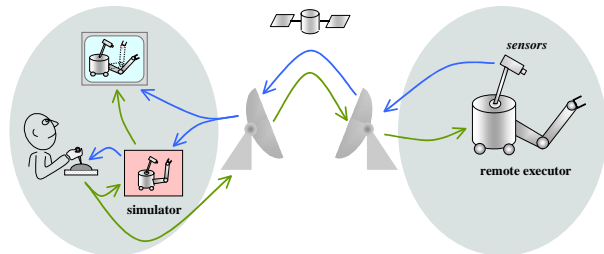


Fig. 2 – Predictive displays

- *Temporal clutch.* This is a modification of the previous strategy [6] that eliminates the need for the simulator to be synchronized with the remote executor allowing the decoupling between the simulator and the remote executor, thus enabling the simulator to run at a higher speed. The operator can decide to enter in this mode and interact with the simulator in order to generate a set of reference points that are later sent to the remote executor which can process them at its highest speed. During the processing of this set of reference points by the remote executor, the operator may engage himself in another interaction with the simulator. A variation of this scheme, called *spatial clutch*, which allows the operator to decouple the simulator from the remote

executor and interact with it for the generation of trajectories requiring a very precise final position has also been proposed. When this desired final position is obtained, a soft transition between the current real position of the arm and the desired final position is automatically generated and the corresponding set of points sent to the remote executor.

- *Teleprogramming*. In this scheme (Fig. 3) the operator interacts with a virtual environment being a set of actions inferred from this interaction [16]. The descriptions of these actions in an appropriate language are sent to the remote executor where they are interpreted and executed. The feedback information is also obtained from the virtual environment but some periodic synchronization with the real world is needed. The remote executor may also have some error recovery capabilities. Since the level of autonomy on the remote executor is increased, virtual environments can be built to allow the operators to work in higher abstraction levels.

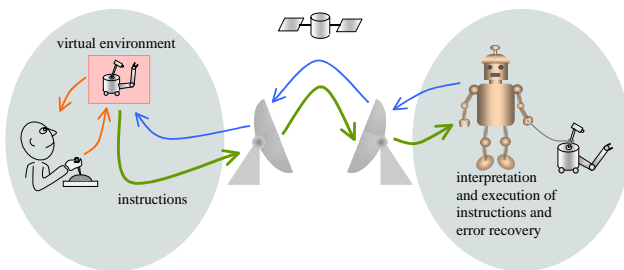


Fig. 3 – Teleprogramming

In all strategies but teleprogramming, the division of functionality between local and remote place is done in a rigid manner, thus sacrificing flexibility.

Teleprogramming is an appealing strategy since it departs from the previous strategies by abandoning the need for this rigid division of functionality between the local and remote places. However, the need for a common language for the specification of the actions to be performed reduces most of its potential interest as a general solution to be used independently of the application domain. This is also true in what concerns the error recovery strategies.

In this work a more general solution is proposed which addresses a wider range of application domains.

III. FITTING AUTONOMY AND MOBILE AGENTS

A rigid distribution of the functionality between the remote and local places is inappropriate because it reduces flexibility. In fact, this solution imposes that the functionality that will ever be needed by all users be known in advance. Besides, it would be much more appropriate if the levels of autonomy could be chosen according to the needs of the application whenever the

characteristics of the communication channels do not impose severe restrictions. For example, some applications require the use of low-level commands for achieving some precision tasks while, in general, low level commanding reduces operator attention.

In the last years the paradigm of mobile agents [13] has emerged. Basically a mobile agent is a program that can migrate from one execution platform to another during its execution. When the agent decides to migrate, its execution state and code are transferred to the destination platform, where the agent is reconstructed and its execution resumed.

When applied for remote operation, this paradigm allows the use of the appropriate levels of autonomy while preserving high levels of flexibility [8][9][11]. In effect, if the remote places are equipped with execution platforms for mobile agents, they have to implement only the low-level commands corresponding to the basic functionality of their resources. In the local places the high-level functionality can be achieved by composing this basic functionality within a mobile agent, which is sent to the remote place where it is then executed almost independently of the characteristics of the communication channels.

In this way the level of autonomy of a remote place can be fitted to the needs of the application and the characteristics of the communication channels (thus the term “fitting autonomy”) [9]. This may require high levels of dynamism in the specification and creation of mobile agents. However the following aspects have to be considered:

- If we consider that in general the agents have to reveal high levels of autonomy (including the capacity to recover from errors and to react to many events) their specification using traditional languages (as those used in most of the traditional mobile agents systems) would be very difficult.
- In many situations the same task defined at appropriate levels of abstraction can be executed using quite different resources. The traditional mobile agents approaches do not consider this aspect and require different agents for each different set of resources that allow the execution of the task.

In this work a solution was adopted which extends the traditional mobile agents paradigm with planning capabilities that allow the agents to instantiate their plans according to the resources they find in each visited place. The agents carry abstract hierarchical plans annotated with information that is used for supervision of their execution.

Fig. 4 depicts the architecture of a mobile agent.

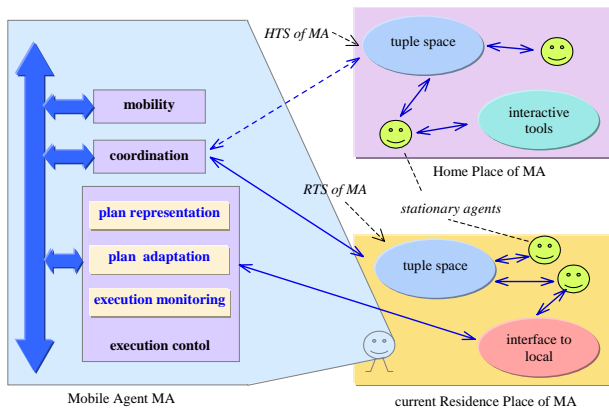


Fig. 4 – General architecture of a mobile agent

In this multi-agent architecture there are several places where mobile agents can reside in each moment of their lives (see Fig. 4). Each place may include several stationary agents, which implement some specific functionality within that place. Stationary agents are involved only in cooperative work with the agents executing in the same place or with mobile agents having this place as home, while inter-place cooperation is achieved using mobile agents. There are several factors motivating this approach, as it will be seen later when coordination is discussed.

It is assumed that all the agents have the same structure, being the only difference between mobile and stationary agents the fact that the latter do not use the mobility capabilities. In fact, besides the homogeneity associated with this solution, it would also allow high levels of flexibility (for example, during the installation of the software component in a new place, or when updating some stationary agent, mobile agents could be launched to the place, adapt their behavior to the specific resources they find and, persistently, spend the rest of their lives there, as if they always were stationary agents).

In the proposed architecture, the main components of a mobile agent are:

- The mobility component that implements the process of agent migration. A platform for execution of mobile agents, called IMAJ (standing for "Intelligent Mobile Agents in Java"), was developed in Java. Due to its multi-platform nature, widespread utilization and richness of built-in services it offers, Java was chosen as the basic development language. However, it lacks symbolic processing capabilities that would reduce programming effort when inherently symbolic processing is necessary (e.g. in planning). This aspect motivated the integration of JINNI (Java INference engine and Networked Interactor [14]) a Prolog-like language interpreter completely developed in Java.
- The coordination component implements the coordination infrastructure that allows the agents to coordinate their activities in order to achieve a well-behaved system. Mobility renders difficult the adoption

of common mechanisms for identification of the localization of the agents. Although tracking mechanisms have been proposed, they carry severe implications on the complexity of the systems and on the execution times. To overcome these difficulties, a coordination model based on Linda-like tuple spaces [17] is adopted. As it can be seen in Fig. 4, a mobile agent has access to two tuple spaces. One of the tuple spaces belongs to the remote place where the mobile agent is executing (RTS - residence tuple space), thus allowing the agent to interact with all the agents that co-reside there (both stationary agents and other mobile agents currently executing in the same place). The other tuple space is merged with the tuple space of the agent's home place (HTS - home tuple space) and allows the agent to interact with the agents located at its home. The part of this tuple space in the agent's side is cached in the current place where the agent is executing and automatically synchronized with the home place whenever a reliable network connection is established. If the agent decides to move to another place and the cache has unsynchronized tuples, they are saved in the agent's state, transferred along with the agent and used to restore the HTS cache when the agent reaches the destination place.

- The execution control component implements the mechanisms that allow one agent to adapt its high level abstract plan to the exact environment it finds in each place it visits, and to execute the adapted plan, including execution monitoring and error recovery. Execution control involves plan adaptation [10], execution monitoring and error recovery [7]. Plan adaptation allows an agent to adapt its high level plan for execution in the current place, according to the capabilities found there; execution monitoring and error recovery is a crucial aspect, since the agents may be operating autonomously for long periods of time. A language called MAAPL (Mobile Agents Abstract Plan Language) [10] was developed for the specification of the plans of the agents. A hierarchical plan structure was considered, since it allows the specification of monitors [15] at various levels of detail, which is very appropriate for complex domains. Furthermore, the hierarchical approach is a powerful mean to structure interesting monitoring strategies. Also, hierarchical plans contribute to the reduction of the complexity of the plan adaptation and error recovery activities.

As can be seen in Fig. 5 a plan has at least one level (level 0). In this level only one action can exist which represents the entire plan. Each action has a refinement attribute that indicates how the action is decomposed at lower levels in the plan hierarchy. For abstract actions that do not exist at the leaves of the abstract plan tree the following refinement types can be specified: *seq* indicating that the action is decomposed into a sequence of other actions, *par*, indicating that the action is decomposed into a set of other actions that can be executed concurrently, and

choice that specifies a set of actions that act as roots of alternative branches at this level.

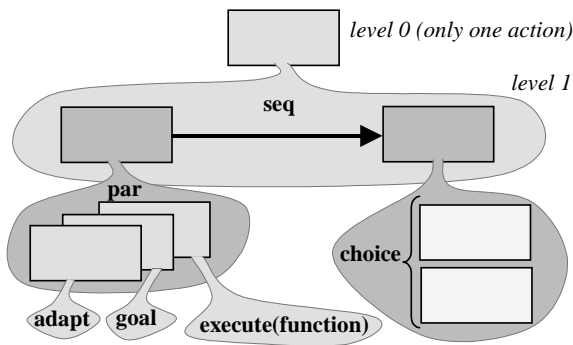


Fig. 5 – Illustration of specification of hierarchical plans

The leaves of the abstract plan must be decomposed in a nonempty set of executable actions. The way this is achieved is indicated by one of the following refinement types: *execute(function)* that allows the user (programmer) to specify a function that implements the functionality associated to the action; and *adapt* and *goal* that specify that a plan must be obtained at the current site taking into account the capabilities of the execution environment. The difference between *adapt* and *goal* is that *adapt* takes the preconditions of the action as the initial state of the planner whereas *goal* takes the current real state of the world. The former is used for obtaining plans which are executed as protocols and so require that execution of the refined plans start with the specified preconditions of the upper level actions satisfied; the latter is used for less restrictive situations where all that is required is the achievement of the effects of the action.

Each action can include a set of monitors that specify some conditions whose violation implies the execution of the specified recovery strategies. Currently the following types of monitors may be specified: *pre-condition monitors* that fire if the specified conditions on the preconditions of the action fail; *effect monitors* fire when the specified conditions on the effects of the action fail; *maintenance monitors* are used for continuous observation of some conditions; *failure monitors* enable the definition of recovery strategies at the parent level when some child action fails; and *time-out monitors* fire if the action does not terminate within the specified time limit.

Each monitor has associated one of the following recovery strategies: *repair*, the default for pre-condition monitors, tries to find a patch plan that repairs the failed conditions; *retry* implies the repetition of the action with its current refinement; *redo* is used to obtain an alternative plan refinement for the action; *ignore* is the default for effect monitors and ignores the error; *fail* is used to propagate the failure to the upper levels in the plan hierarchy; *user(function)* that allows the programmer to specify his own strategy.

Another important aspect while considering real-time response is that the execution of one task must not cause significant degradation of the execution times of the other tasks, which turns the sequential execution approach found in many systems inappropriate [7]. This is the reason why the architecture of the execution supervision component (Fig. 6) extensively uses multi-thread programming.

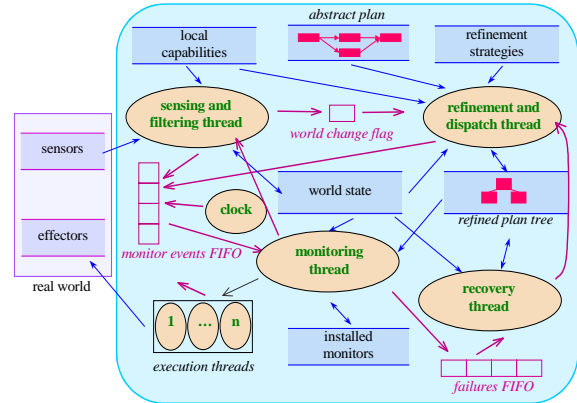


Fig. 6 - Architecture of the execution supervision component of a mobile agent

This means that, provided enough resources are available, parallel branches in a plan are executed in parallel, because, besides other concurrent activities, each primitive action runs in its own execution thread. Furthermore, multi-threading simplifies the implementation of reactive behaviors related to the events that occur during the execution of an action.

In this architecture, executable actions (EA) are performed as concurrent threads and interface with the local environment via a set of effectors. They call local procedures determined by the capabilities they are associated with. It is admitted that the local execution environment is equipped with adequate monitoring and error recovery mechanisms at the execution level, in such a way that if execution of an EA fails, sufficient information for characterizing the failure is obtained from the local environment. This information is used by the high-level error recovery.

The sensing and filtering thread is a high priority thread that runs periodically and is responsible for maintaining an updated persistent world model, at a rate determined by the dynamics of the domain. A notification event is sent to the monitoring thread (via the monitor events FIFO) for each perceived change of the world state. The refinement and dispatch thread is also notified of these changes (via the world change flag - a binary semaphore).

The refinement and dispatch thread runs in two phases: i) in the first phase it adapts the carried abstract plan; ii) in the second phase it acts as a dispatcher, scheduling next EAs for execution. When all pending EAs are waiting for the completion of precedent actions, this thread blocks in

the world change flag to wait for a world change (the changes in the state of an action are also seen as a world change).

The monitoring thread handles all the events in the agent (change of the world state, clock ticks, start and termination of EAs, etc.). It installs and removes monitors when EAs start and terminate, and checks the installed monitors when clock events or world change events are received. When a monitor condition is violated, the recovery thread is notified via the failures FIFO.

The recovery thread executes the recovery procedures associated to the notified error, trying to obtain a suitable plan repair that hopefully will allow the normal continuation of the agent's execution.

IV. AN APPLICATION

The proposed solution has been evaluated in a real scenario comprising the remote manipulation of an assembly robot through the Internet. As Fig. 7 depicts, the remote place is equipped with a Scara-type robot, a set of three grippers, a pallet that supports the assembly of the product and two dispensers (one for parts of kind A and the other for parts of kind B).

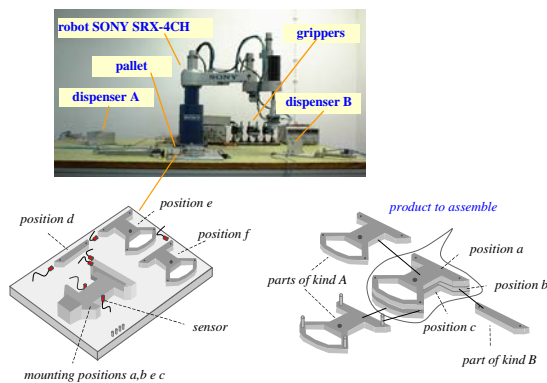


Fig. 7 – The configuration of the remote place

The original idea of this environment was the experimentation with a simplified version of the Cranfield benchmark, but in our case we assume that remote users may be interested in different experiments using the basic scenario. For instance, some users may be interested in assembling the product as shown in Fig. 7, whereas others may be interested in disassembling the product or whatever can be done with this scenario.

The remote place and the local places are equipped with execution platforms for mobile agents. In the remote place only the basic functionality of the existing resources is available. For example, for the robot, this functionality is implemented by the robot controller (*move*, *get gripper*, *open gripper*, etc.). This allows users to choose the more appropriate level of abstraction for their

applications. There are two cases to consider in this respect:

- It is necessary that the relevant information about the state of the remote place be present in the local places. But the state information in the remote place is naturally of a low level nature that, in general, is distant from the levels of abstraction more appropriate for the applications running in the local places.
- From the operational point of view it is more adequate for the users to specify tasks using symbolic domains and high abstraction levels that may differ in a great extent from the low level operational characteristics available in the remote places.

For the first case, the solution that would immediately come into mind would be to gather the low level sensorial information available in the remote place and send it to the local places. However this would require the transference of higher quantities of information than the necessary if only the relevant information described at the levels of abstraction used by the applications in the remote places was transferred. We have adopted a solution in which specific status gathering agents (*remoteFdBkAg* in Fig. 8) are sent from the local places to the remote place. These agents filter the gathered information in the remote place, convert the filtered information to the levels of abstraction of the associated applications and send the results to the local places. This is achieved by making each application launching its remote feedback agent in the respective local place. The agent is prepared to move to the remote place as its first task. In this way the agent has access to the tuple space of the respective local place (its home place) and can use it to send information to there. In each local place other agents can get this information and process it according to the needs of the application. For example, in the case of Fig. 8, the agent *localFdBkAg* gets this information from the tuple space and shows it in a graphical interface.

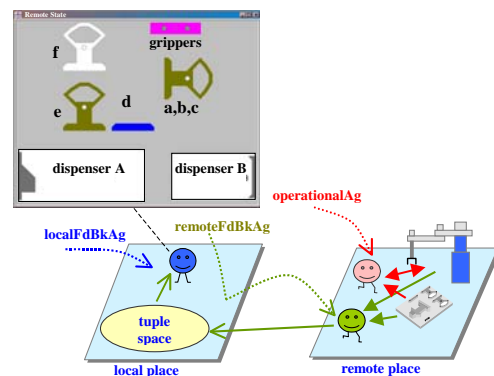


Fig. 8 – Illustration of the adopted remote operation strategy

The second case is handled by allowing the users to specify operational agents at the desired abstraction level. These agents are sent to the remote place where they execute taking into account the basic functionality of the existing resources (as is the case of the agent

operationalAg in Fig. 8). This is possible due to assumptions that a common ontology for the specification of the capabilities used in the remote places is available and that the agents are equipped with planning capabilities which allow them to obtain an executable set of actions based on the abstract plan they carry and on the set of capabilities available in each remote place they visit.

Among the various possible experiments based on this scenario and using a task level control strategy, the following two are considered in this paper:

A. Experiment 1 – Influence of time-delays on the completion-times of the tasks

In this experiment a complete assembly of one product was done using two approaches. In the first approach the assembly was done using the traditional remote procedure calling mechanism under a simulated 10 second round-trip time-delay and using low level commands such as *move*, *open gripper*, *get gripper*, *close gripper*, etc. The task was terminated after about 18 minutes. A second experiment was done using a mobile agent that was launched in the remote server where it did the same assembly task spending about 3 minutes (including transfer time of the agent and the time it spent in obtaining the executable plan). These results confirm that remote operation using low-level commands is impracticable when the communication channels have large time-delays. One alternative would be to increase the level of the commands implemented in the remote server, but this solution sacrifices the flexibility of the system, as it was discussed previously. The use of mobile agents is a good solution, as long as they are provided with high levels of autonomy.

B. Experiment 2 – Autonomy of the mobile agents

In this experiment a maintenance procedure over an already existing product was considered. It was assumed that the parts of kinds *A* and *B* have to be submitted to some treatments carried out on positions *f* and *d*, respectively. This means that the part of kind *A* on position *a* has to be moved to position *e*, the part of kind *B* on position *b* has to be moved to position *d*, then the part of kind *A* on position *c* is moved to position *f* where it is submitted to the considered treatment and moved again to position *c*. After that, the part on position *d* can be moved to position *b* after which the part of kind *A* at position *e* is moved to position *f* before its movement to position *a*.

Since the low-level functionality provided by the robot controller is inappropriate for achieving an easy way to specify the missions of the agents, an abstract symbolic model of the world was defined. In general, this definition would be done once by a domain expert who would also specify procedures that the agents will use to convert the

low-level sensorial information to the high-level abstract conditions and to execute the actions specified in the abstract domain (which may require the execution of many low-level orders for the used resources). The users specify the missions of the agents using this abstract world. In the case of this experiment an abstract model was defined which considers the conditions defined in TABLE I.

TABLE I
ABSTRACT CONDITIONS

Condition	Meaning
mounted(P)	a part is mounted on position <i>P</i>
confirmed(P)	placement of a part on position <i>P</i> (<i>c</i>) is confirmed
freeA(P)	position <i>P</i> (<i>e</i> or <i>f</i>) for parts of kind <i>A</i> is free
freeB(P)	position <i>P</i> (<i>d</i>) for parts of kind <i>B</i> is free
free(P)	mounting position <i>P</i> is free
existsA(P)	part of kind <i>A</i> available at position <i>P</i>
existsB(P)	part of kind <i>A</i> available at position <i>P</i>
availableA	part of kind <i>A</i> available in dispenser <i>A</i>
actuatedA	dispenser <i>A</i> is working
availableB	part of kind <i>B</i> available in dispenser <i>B</i>
actuatedB	dispenser <i>B</i> is working

The set of planning operators shown in TABLE II was also defined. They are used for instantiation of the abstract plan on the remote site and also for error recovery.

TABLE II
PLANNING OPERATORS

Operator	Function
mountA	mount part of kind <i>A</i> on position <i>a</i> , from position <i>f</i>
dismountA(P)	dismount part of kind <i>A</i> from mounting position <i>a</i> to position <i>P</i> (<i>e</i> or <i>f</i>)
mountB	mount part of kind <i>B</i> on position <i>b</i> , from position <i>d</i>
dismountB(P)	dismount part of kind <i>B</i> from mounting position <i>b</i> to position <i>P</i> (<i>d</i>)
mountC	mount part of kind <i>A</i> on position <i>C</i> , from position <i>f</i>
dismountC(P)	dismount part of kind <i>A</i> from mounting position <i>c</i> to position <i>P</i> (<i>e</i> or <i>f</i>)
moveEF	move part of kind <i>A</i> from position <i>e</i> to position <i>f</i>
placeA	place part in the output of dispenser <i>A</i>
moveA(P)	move part from the output of dispenser <i>A</i> to position <i>P</i> (<i>e</i> or <i>f</i>)
placeB	place part in the output of dispenser <i>B</i>
moveB(P)	move part from the output of dispenser <i>B</i> to position <i>P</i> (<i>d</i>)
trash	remove part of kind <i>A</i> from mounting position <i>c</i> and discard it

It is worth to mention again that the execution of an action derived from a planning operator may require the execution of many commands at the resource level.

As it can be seen in TABLE II, the operator *trash* discards the part on mounting position *c*. This operator was included because the action of placing a part of kind *A* on position *c* is prone to errors due to the required precision that cannot be achieved in many situations due to small slipping of the part in the gripper.

The definition of each operator includes the pre-conditions for application of the operator, the expected effects, and also the set of capabilities that must exist in the current execution environment in order that the operator can be considered by the planning system.

Based on the abstract model described above, the maintenance agent can be specified using the abstract plan shown in Fig. 9, where the MAAPL code for the agent is only partially presented due to lack of space.

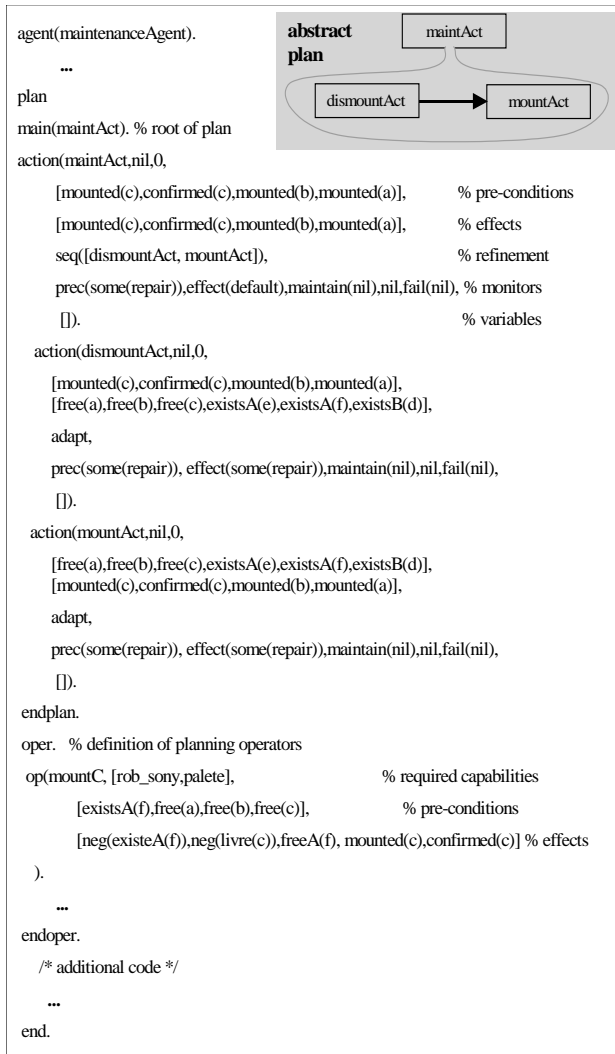


Fig. 9 – Specification of the maintenance agent

The action named *maintAct* has the same set of conditions on its pre-conditions and effects, which makes sense since we want the product to be assembled after the maintenance procedure. This set specifies that the product be completely assembled. Action *dismountAct* specifies in its pre-conditions that the product must be completely assembled and in its effects that the parts of kind *A* must be on positions *e* and *f* and that the part of kind *B* must be on position *d*. This ensures that after the execution of this action the first part of kind *A* may be mounted on position *c* from position *f*. Action *mountAct* has as pre-conditions the same set of conditions specified in the effects of the previous action and as effects the conditions that require that the product be completely assembled. Since the operators for mounting parts of kind *A* require that the part to be mounted is on position *f* and the operator for mounting parts of kind *B* require that the part is on position *d*, the abstract plan is a correct specification of the maintenance procedure.

We could include in the abstract plan another level composed only of executable actions (corresponding to instances of the operators) but this is not necessary since the agent can do this by itself using its planning capabilities.

When one agent with this abstract plan was sent to the remote place, it generated the executable plan shown in Fig. 10.

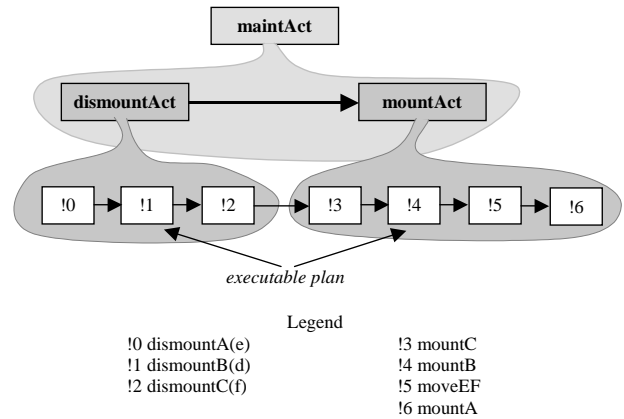


Fig. 10 – Result of plan adaptation for the plan of Fig. 9

During the execution of this plan several errors were provoked. The agent recovered from the errors whenever it was theoretically possible. Among the several cases studied the following were selected for this paper:

Case 1:

Immediately before the start of action !6 an error was provoked which consisted in removing the part of kind *A* from position *f*. This caused the firing of the pre-condition monitor of action !6 whose default strategy is *repair*. The agent obtained a repair plan to eliminate this failure as is shown in Fig. 11, where only the executable actions are shown. Action !7 puts one part of kind *A* in

the output of the dispenser *A* and action !8 moves this part to position *f*, which makes possible the resuming of the initial executable plan.

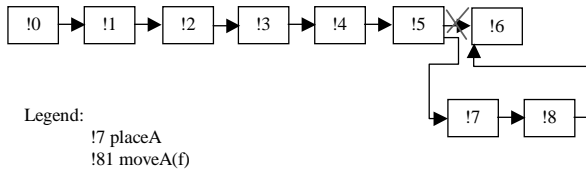


Fig. 11 – Result of plan repairing (case 1)

Case 2:

The execution of action !3 failed due to slipping of the part on the gripper. Therefore condition *confirmed(c)* was not achieved. Since the effect monitor for the executable action is *ignore*, which means that the agent proceeds the execution of the plan without taking into account this error, the error propagated to the pre-conditions of action !4. Again a repairing plan was obtained as indicated in Fig. 12. This repairing plan makes the agent discard the part whose presence is not confirmed and to get another part from dispenser *A*, which is moved to position *f* before being mounted.

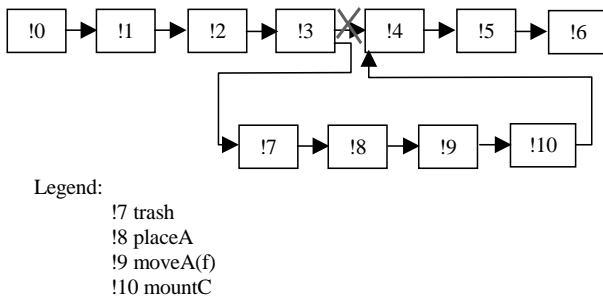


Fig. 12 – Result of plan repairing (case 2)

In this example it was assumed that during plan repair the agent tries to preserve all the pre-conditions of other actions awaiting for execution that are not fulfilled by any other action in the plan that executes before them and also the effects of the plan that are not fulfilled by any other action not yet executed. This strategy avoids propagation of errors, but may impose larger recovery times. It is also possible to follow an optimistic strategy under which only the failed conditions are considered for plan repairing. Fig. 13 depicts what would occur if this strategy were used. As it can be seen, the error in the condition *confirmed(c)* causes the repairing plan marked with “1”. This plan moves the part causing the error to the trash, moves the part present in position *e* to position *f* and then places this part in position *c*. But this plan later originates an error in the pre-conditions of action !5 which are repaired by the plan marked with a “2” in Fig. 13.

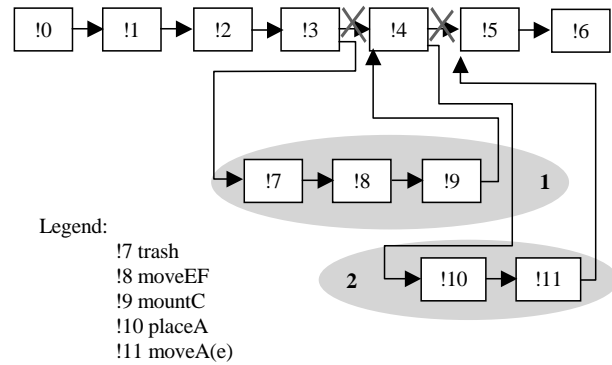


Fig. 13 – Illustration of the results of plan repairing when an optimistic approach was adopted

V. CONCLUSIONS

In this paper the concept of fitting autonomy in remote operation was presented which considers the definition of the levels of autonomy of the remote executors according to the needs of the applications and the characteristics of the communication channels.

Mobile agents were proposed as a way to implement systems with fitting autonomy, since they make possible the conciliation of high levels of flexibility with high levels of independence on the characteristics of the communication channels. However, the agents have to be provided with high levels of autonomy, which implies the need for execution monitoring and error recovery, which traditionally have been tackled with specific solutions for specific application domains.

An architecture for mobile agents intended for use in remote operation without any compromise with specific application domains was described. The architecture includes the components of mobility, communication/coordination among agents, and execution control. The specification of the agents is done by means of hierarchical abstract plans annotated with information that guides the supervision of the execution of the agent. This abstract plans are instantiated in each site visited by the agent according to the available capabilities. A language (called MAAPL) is used to specify the abstract plans. Its high level and essentially declarative nature eases the process of plan specification and maintenance.

The use of this architecture for the control of a robot through the Internet was described. The achieved results confirm the claimed advantages of the proposed solution.

VI. ACKNOWLEDGMENTS

This work was supported in part by the Portuguese Ministry of Science and by the European Commission through the TeleCARE project.

VII. REFERENCES

- [1] K. Taylor, "Issues in Internet Telerobotics", *International Conference on Field and Service Robotics*, Canberra, Australia, 1997.
- [2] A. Dutta-Roy, "Networks for Homes", *IEEE Spectrum*, December 1999, pp. 26-33.
- [3] Sun: Sun Microsystems and Whirlpool Corporation team up to build networked home solutions, Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, 2000, USA.
- [4] T. B. Sheridan, "Teleoperation, Telepresence, and Telerobotics", *Human Factors in Automated and Robotic Space Systems: Proceedings of a Symposium*, T. B. Sheridan, D. S. Kruser and D. Deutsch eds, Washington, D.C. National Research Council, 1987, pp. 279-291.
- [5] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*, The MIT Press, ISBN 0-262-19316-7, 1992.
- [6] L. Conway, R.A. Volz, M.W. Walker, "Teleautonomous Systems: Projecting and Coordinating Intelligent Action at a Distance", *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 2, 1990, pp. 146-158.
- [7] W.J. Vieira, L. M. Camarinha-Matos, "Execution monitoring in adaptive mobile agents", *Proceedings of CIA'99 - Third International Workshop on Cooperative Information Agents*, Lecture Notes in Artificial Intelligence 1652 (Springer), Uppsala, Sweden, ISBN 3-540-66325-8, 1999, pp. 220-231.
- [8] W.J. Vieira, L.M. Camarinha-Matos, "Adaptive Mobile Agents: Enhanced Flexibility in Internet-Based Remote Operation", *Proceedings of the 4th IFIP/IEEE International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Transportation (BASYS'2000)*, in L. M. Camarinha-Matos, H. Afsarmanesh and H. Erbe (Ed.), *Advances in Networked Enterprises, Virtual Organizations and Systems Integration*, Kluwer Academic Publishers, 2000, pp. 161-174.
- [9] W.J. Vieira, *Adaptive Mobile Agents for Remote Operation*, PhD Thesis, New University of Lisbon, 2001 (in Portuguese).
- [10] L.M. Camarinha-Matos, W.J. Vieira, "MAAPL: A language for adaptive mobile agents with execution monitoring". *Proceedings of INES'99, 3rd IEEE International Conference on Intelligent Engineering Systems*, Poprad, High Tatras, Stará Lesná, Slovakia, ISBN 80-88964-25-3, 1999, pp. 171-176.
- [11] L.M. Camarinha-Matos, W.J. Vieira, L.O. Castolo, "A Mobile Agents Approach to Virtual Laboratories: Enabling remote operation over the Internet", *Proceedings of the IFAC Symposium on Artificial Intelligence in Real Time Control (AIRTC'2000)*, Budapest, Hungary, 2000, pp. 301-306.
- [12] D. Bernard, G. Dorais, E. Gamble, B. Kanefsky, J. Kurien, G. Man, W. Millar, N. Muscettola, P. Nayak, K. Rajan, N. Rouquette, B. Smith, W. Taylor, Y. Tung, "Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment", *Proceedings of the American Institute of Aeronautics and Astronautics Space Technology Conference*, 1999.
- [13] J. White, "Mobile Agents", in J. Bradshaw (Ed.), *Software Agents*, AAAI/MIT Press, 1997, pp. 437-472.
- [14] Jinni, Java Inference Engine and Network Interactor, <http://www.binnnetcorp.com/Jinni/index.html>.
- [15] G. Reece, A. Tate, "Synthesizing Protection Monitors from Causal Structure", *Proceedings of the second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 1994.
- [16] C. Sayers, *Remote Control Robotics*, Springer-Verlag New York, ISBN 0-387-98597-2, 1999.
- [17] G. Papadopoulos, "Coordination Models and Languages", in *Advances in Computers*, Vol. 46: *The Engineering of Large Systems*, Academic Press, 1998, pp. 329-400.