



A Mobile Agents Approach to Virtual Laboratories and Remote Supervision

L. M. CAMARINHA-MATOS and OCTAVIO CASTOLO

*New University of Lisbon, Faculty of Sciences and Technology, Quinta da Torre,
2829-516 Monte Caparica, Portugal; e-mail: {cam,lgc}@uninova.pt*

WALTER VIEIRA

*Instituto Superior de Engenharia de Lisboa, DEEC, Portugal; and New University of Lisbon,
Faculty of Sciences and Technology, Quinta da Torre, 2829-516 Monte Caparica, Portugal;
e-mail: wv@uninova.pt*

(Received: 4 April 2001; in final form: 17 October 2001)

Abstract. This paper presents the use of adaptive mobile agents for remote operation, enabling real-time response in spite of the limitations of the communication channels in terms of time-delays, availability, and reliability. Autonomy of the mobile agents is achieved through high levels of intelligence including execution monitoring and error recovery. Potential applications range from traditional telerobotics to virtual laboratories where mobile agents act as representatives of users in scientific experiments. Practical results are presented in a scenario where a SCARA-type robot is remotely commanded through the Internet.

Key words: mobile agents, teleoperation, remote supervision.

1. Introduction

Virtual laboratories, tele-operation and remote supervision in general, are attracting growing interest, due to the many potential applications, including areas such as sharing expensive equipment, control of machines operating in hazardous or inaccessible environments, cooperative supervision of manufacturing processes in virtual enterprises, and spatial vehicles operating with large autonomy. In special, the number of applications using remote operation over the Internet has been growing at significant rates in various domains, such as remotely operated robots and telescopes, manufacturing systems, remote elderly care, remote surveillance systems, etc.

In this paper the specific case of the Internet, whose low costs and widespread availability make it very appealing as a basis for remote operation, is considered. The proposed approach addresses some difficulties, such as:

- (i) High levels of heterogeneity can be expected in the variety of sensors and devices to be found in the remote places when reasonable practical application domains are considered, which can degrade the flexibility and scalability of the system;

- (ii) Internet is still characterized by long and variable time-delays and, very often, suffers from low levels of availability, raising new challenges in what concerns the reliability of the implemented system and its dependence on the characteristics of the network; and
- (iii) The remote execution environments are potentially uncertain which means that it is not adequate to resort to deterministically programmed systems. Therefore, special attention is given to the aspects related to the large time-delays and the poor availability of the network connections. Related aspects, such as heterogeneity and uncertainty of the execution environments, which have strong impact on the systems design, are also addressed.

The proposed approach to enable remote operation and supervision of devices over communication channels with large time-delays and/or poor availability is based on mobile agents [7, 8, 13] with adaptive behavior that carry hierarchical high level abstract plans which they adapt to the actual capabilities of each visited remote place [4]. The hierarchical plans are annotated with information intended to guide the plan adaptation, execution monitoring, and error recovery processes. The mobile agents are supported by a multi-thread architecture, which is a fundamental characteristic for the implementation of the necessary concurrency between plan execution and monitoring.

The virtual lab application can also benefit from the autonomy of the agents in order to not require a synchronous availability of the participants in a given experiment. These participants can delegate on their agent representatives the actual realization of some task, which will be done when the necessary conditions are satisfied. This approach allows high levels of decoupling between the participants in the experiment, both in spatial and temporal terms, since they do not need to be physically present in the place where the experiment takes place and neither do they need to participate on-line in the experiment.

This (multi) mobile-agent approach represents a sound basis for a flexible implementation of remote manipulation or virtual laboratory systems. Potential applications are in providing remote access to experiments in collaborative research, specially, in the case of expensive setups only available in a few places. Another application area is remote learning, especially for life-long training programs. Remote access to virtual teaching labs gives the student the opportunity to get almost “hands on” experience.

The issues related with plan adaptation were first discussed in [4]; execution monitoring and error recovery were presented in [11]; and the language MAAPL (Mobile Agents Abstract Plan Language) for description of hierarchical abstract plans with execution monitoring was described in [5]. An integrated and detailed view of these issues can be found in [10]. In this paper an implemented laboratorial experiment is described focusing on the error monitoring and recovery aspects, and a partial assessment of results as well as an identification of further research challenges is made.

2. The Internet and Remote Operation

Current Internet infrastructures put severe constraints to real-time operation, mainly due to:

Large time-delays. Various authors have addressed this problem, and solutions have been proposed for very specific domains falling into two main classes. In one class the time-delay is estimated and somehow presented to the operator. For instance, by estimating the time-delay and, through simulation or other visualization methods, giving the operator an indication of the expected status of the remote place. For instance, in telerobotics applications, a cursor may be moved over a video image of the remote scene to point to the expected new position of the robot arm after a “*move*” command. In the second group, the intelligence and autonomy of the remote place is increased in order to provide local high-level supervisory control strategies in the sense that the operator only gives high level commands to the remote place when the remote place cannot proceed by its own means. The first case is only applicable to very specific domains whereas the second reduces the dependency on the characteristics of the network at the expenses of the flexibility, since any change of functionality or the alteration of the remote environments require a software update of the remote places. The solution described in this paper is based on mobile agents that are sent to the remote place to implement some desired functionality. Since the agents run on the remote place, high levels of independence on the characteristics of the network are achieved, yet preserving high levels of flexibility (new mobile agents can be built whenever needed to accomplish the desired functionality).

Poor availability of the network. This situation demands high levels of autonomy in the remote place that should be provided with skills to allow local execution supervision and to adopt intelligent recovery strategies when errors occur. Some authors have tried to attack this problem with full deliberative agents [14] equipped with planning capabilities that allow them to re-plan the new sequence of actions when errors occur. In spite of the recent developments in automated planning [12], this is impracticable for many real domains due to performance insufficiency or to the complexity of the considered domain, which makes difficult the implementation of full autonomous systems. In our proposal the mobile agents carry hierarchical abstract plans (a mission) annotated with monitoring and error recovery information. These annotations and the hierarchical abstract plan are used as guidelines by the execution supervision component of the agents. As the abstract plans may be generated off-line by mixed initiative planners, it is possible to embed very sophisticated execution and error recovery strategies. On the other hand, the hierarchical structure of the plans allows an easier control of the complexity of plan repair during error recovery.

Heterogeneity of the execution environments. This is a crucial aspect when considering remote operation in large scale involving distributed and autonomous fa-

cilities. It is not practical to have a different agent for each differently equipped remote place, even in those cases where the desired functionality at some reasonable abstraction level is the same. Furthermore, due to remote node's autonomy, it is not always possible to be aware of local changes. In this work the abstract plans carried by the agents are adapted (i.e., refined) according to the actual capabilities the agents find in the remote places they visit. As in the previous point, the hierarchical nature of the abstract plans is the key to control the complexity of the plan adaptation. This refinement capability is, of course, limited to a target domain for which the remote agent was designed.

Although the Internet2 promises to solve some of the above problems, some difficulties still exist:

1. It is not likely that widespread access will be guaranteed in the short term, at least outside the education and research institutions.
2. Problems related to failures will ever exist.
3. It is not clear how the increase of the bandwidth of the network can follow the demands of network traffic made by more and more network centric applications.
4. There are some domains where Internet2 will not bring significant benefits. For example, mobile users connected to the network through wireless channels will continue to experience low-levels of availability of the network connections.

3. An Architecture for Adaptive Mobile Agents

In order to cope with the mentioned characteristics, an architecture for adaptive mobile agents was designed by the Robotics and Integrated Manufacturing Group of the New University of Lisbon. It is called IMAJ (Intelligent Mobile Agents in JAVA) and comprises three main components (Figure 1):

1. The mobility component that implements the process of agent migration (see [3] for more details on this component).
2. The coordination component that implements the coordination infrastructure allowing the agents to coordinate their activities in order to achieve a well behaved system. Basically, each agent communicates with the other agents currently executing in the same server through a local tuple-space. Each agent also sees the tuple-space of its home place (the server where it was launched for the first time) through which it can cooperate with agents originated in the same server [2].
3. The execution supervision component that implements the mechanisms allowing one agent to adapt its high level abstract plan to the specific environment it finds in each visited place, and to execute the adapted plan, including execution monitoring and error recovery [11].

Execution supervision involves plan adaptation, execution monitoring, and error recovery. Plan adaptation allows an agent to adapt its high level plan for execution in each place, according to the capabilities found there; execution monitoring and

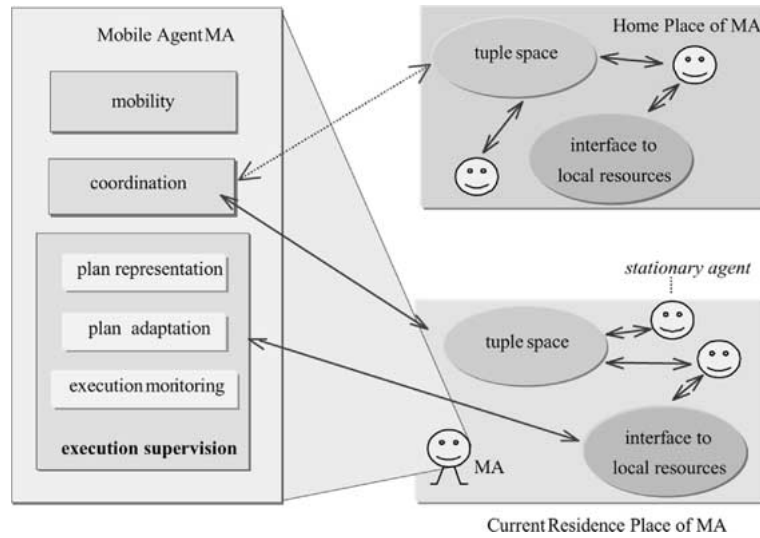


Figure 1. Mobile agent's structure.

error recovery are crucial since the agents may be operating autonomously for long periods of time.

In contrast with most previous works on execution supervision that were focused on very restricted application domains, this work is concerned with more general solutions that address a wide range of application domains implying that the actual composition of each execution environment cannot be known in advance. Furthermore, when dealing with remote operation, the agents must run with a high degree of autonomy in uncertain environments. To achieve this goal, an approach based on general monitoring and recovery methods and on plans with annotations intended to help the execution monitoring and error recovery was adopted. The applied hierarchical plan structure allows the specification of monitors at various levels of detail, which seems quite appropriate for complex domains. Furthermore, the hierarchical approach is a powerful mean to structure interesting monitoring strategies that range over a set of low level actions.

Another important aspect when considering real-time response is that the execution of one task should not cause significant degradation of the execution of other tasks, which turns the sequential execution approach found in many systems inappropriate. This is the reason why the architecture of the execution supervision component (Figure 2) extensively uses multi-thread programming. This means that provided enough computational resources are available, parallel branches in a plan are executed in parallel. Each primitive action runs in its own execution thread, which simplifies the implementation of reactive behaviors according to the events that occur during the execution of an action.

In this architecture, the executable actions (EA), performed as concurrent threads, interface the local environment via a set of effectors. They call local procedures or

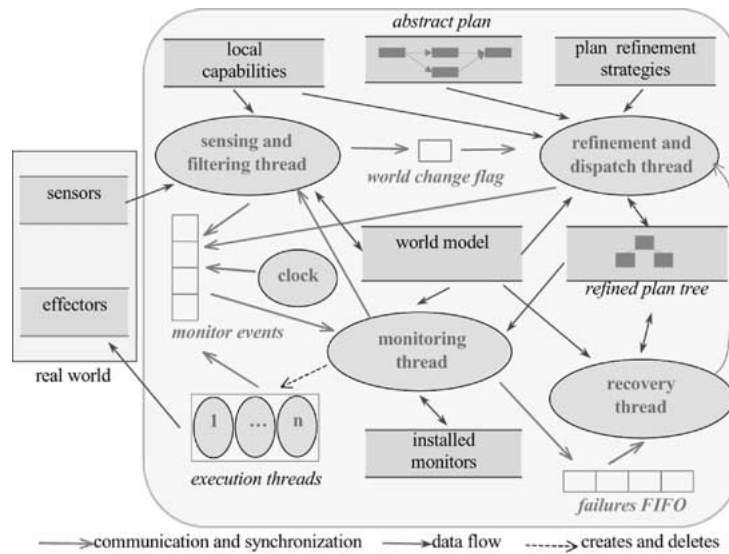


Figure 2. Architecture of the execution supervision component of a mobile agent.

services 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 the execution of an EA fails, sufficient information for characterizing the failure is obtained from the environment. This information is used for high-level error recovery.

The sensing and filtering thread is a high priority task 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 monitoring thread handles all the events in the agent (change of the world state, clock ticks, start and termination of EAs). 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 by 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.

<pre> action(mount_bell, nil, 0, [free(a),free(b),free(c)], [mounted(c),confirmed(c),mounted(b), mounted(a)], seq([pieces_on_pallet,mount_pieces]), prec(some(repair)), effect(some(retry)), maintain([]), nil, fail([]), []). </pre>	<pre> action name parameters priority pre-conditions effects refinement (set of partial ordens) pre-condition monitors effect monitors maintance monitors time-out monitors fail monitors variables </pre>
<pre> action(pieces_on_pallet, nil, 0, [freeA(e),freeA(f),freeB(d)], [existsA(e),existsA(f),existsB(d)], adapt(strips), prec(some(repair)), effect(some(retry)), maintain([]), nil, fail([]), []). </pre>	<pre> action name parameters priority pre-conditions effects refinement pre-condition monitors effect monitors maintance monitors time-out monitors fail monitors Variables </pre>
...	

Figure 3. Excerpt of a plan specification in MAAPL.

The specification of the abstract plans carried by the agent is done through the language MAAPL (Mobile Agents Abstract Plan Language) [5]. As mentioned above, this language allows the definition of hierarchical abstract plans with the specification of several monitors [9] and several recovery strategies if a monitor fires. The specification form of an abstract plan is shown in Figure 3. Basically, the specification of each action is based on the STRIPS model for definition of planning operators [6]. As Figure 3 illustrates, each action is composed of a name, a set of parameters, a number indicating the priority of the action for run-time decision in case of concurrency for resources, a set of pre-conditions which are predicates that indicate some conditions in the world that have to be satisfied for the start of the action's execution, a set of effects which are predicates that indicate that some conditions in the world must be achieved by the execution of the action, a refinement attribute which indicates how the action is decomposed in the lower level of the hierarchical planning (in the case of Figure 3 action `mount_bell`'s refinement indicates that it is decomposed in a sequence of two other actions: `pieces_on_pallet` and `mount_pieces`), a set of monitors which allow the definition of recovery strategies that will be executed if some events occur during the actions' execution, and a set of variables which can act as temporary storage locations during the actions' execution.

Monitors are very important elements in the execution supervision. Currently the following types of monitors can be specified:

- *Pre-condition monitors* that fire if the specified condition on the preconditions of the action fails;
- *Effect monitors* that fire when the specified condition on the effects of the action fail;
- *Maintenance monitors* that are used for continuous observation of some condition;
- *Failure monitors* that enable the definition of recovery strategies at the parent level when some child action fails; and
- *Time-out monitors* that fire if the action does not end within the specified time limit.

One of the following recovery strategies is associated to each monitor:

- *repair* is the default for pre-condition monitors and 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* defines an user function for recovery the error.

The hierarchical refinement in MAAPL can use one of the following mechanisms:

- *seq* that specifies a sequence of subactions;
- *choice* that specifies a set of actions one of which must be chosen;
- *par* that specifies a set of actions whose refinements will be executed concurrently;
- *execute* that specifies a set of instructions for action's tasks defined by the user, and
- *adapt* and *goal* that specify that a plan must be obtained locally taking in account the capabilities of the environment.

The difference between *adapt* and *goal* is that *adapt* takes, during the plan refinement phase, the preconditions of the action as the initial state of the planner whereas *goal* takes, during the execution of the plan, the current real state. The former is used for obtaining plans that are executed as protocols and so require that execution of the refined plan starts at the preconditions of the upper level action; the latter is used for less restrictive situations where all that is required is the achievement of the effects of the action. With *adapt* it is possible that during the execution of the refined plan the real world state does not match the pre-conditions of the abstract action which were used by the planner to achieve the refined plan. In such cases a pre-condition failure is generated and the corresponding recovery strategy executed.

The executable actions of a task are associated to the planning operators. The execution environment could provide these operators but it seems more reason-

Operator	Capabilities	Pre-conditions	Effects
mountA(P) <i>mounts part of type A on position a from position P</i>	rob_sony pallete	existsA(P) free(a) mounted(b)	¬existsA(P) ¬free(a) freeA(P) mounted(a)

Figure 4. A service operator.

opt mountA(P)	operator name
[rob_sony, pallete],	required capabilities
[existsA(P), free(a), mounted(b)],	pre-conditions
[neg(existsA(P)), neg(free(a)), freeA(P), mounted(b)],	effects
).	

Figure 5. Excerpt of an operator definition in MAAPL.

able that the agent itself defines them, because this allows the agents to use the abstraction levels that are more reasonable for the application. The execution of the actions corresponding to the planning operators uses a standard set of services for each device that are supported by the execution environment. Figure 4 shows a planning operator with its own pre-conditions and effects.

Figure 5 shows an example of how to model service operators in MAPPL. As Figures 4 and 5 show, each operator is composed of a name, a specification of the required capabilities (resources on workshop) to perform it task, a set of pre-conditions with are predicates that indicate some conditions in the world that have to be satisfied for executing the task, and a set of effects which are predicates that indicate some conditions in the world that will be achieved by the execution of the operator's task action.

4. An Evaluation Scenario

4.1. THE SCENARIO

For evaluation of the proposed solution an assembly scenario was considered in which a simplified version of the Cranfield benchmark (assembly of a pendulum) is performed. The experiments consist of assembling three parts, as in Figure 6, using a task-level control strategy. The remote place environment is composed of a SCARA-type robot (SONY SRX-4CH), a pallet for assembly, and two dispensers, one for parts of type *A* and the other for parts of type *B*. This environment is enriched with a set of infrared sensors for detecting the presence of parts in several points, as shown in Figure 6. An inexpensive video camera is also included, allowing visual feedback from the remote site to be provided at the supervision side.

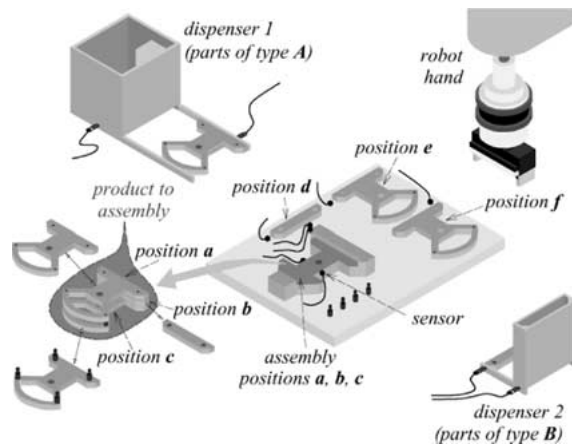


Figure 6. The remote environment layout.

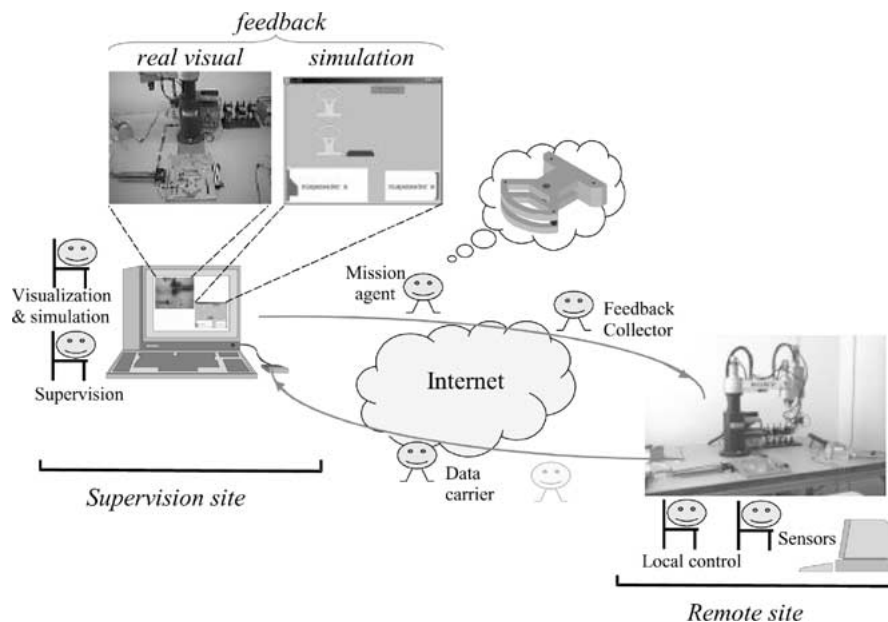


Figure 7. Feedback collector and visualization agents.

An IMAJ server is integrated in this remote site environment. A second IMAJ server is installed in another machine (supervision site) that communicates with the first server through the Internet. In this second server a client environment is set up to receive sensorial feedback from the remote server and to allow the user to launch mobile agents. A *Feedback Collector* agent is launched in the supervision site server and immediately migrates to the remote site server where it continually senses the environment and puts the sensed information in its home tuple-space. A *Visualization and Simulation* agent runs in the supervision site server and dis-

plays the information sent by the *Feedback Collector*, as illustrated in Figure 7. In addition to these, other agents may be launched as, for instance, an agent for local control in the remote station, a supervision agent at the supervision site, etc.

4.2. TYPES OF EXPERIMENTS

Two types of experiments were conducted:

4.2.1. *Implications of Time Delays*

Since the various experiments were conducted using a task-level control strategy, the main implication of the time-delays is on the time taken to complete the assembly task.

EXPERIMENT 1. In a first experiment the assembly task was performed using the traditional remote procedure call mechanism under a (simulated) 10 second round-trip time-delay and using low level commands such as *move*, *open grip*, *get grip*, *close grip*, etc. The task was terminated after about 18 minutes.

EXPERIMENT 2. A second experiment was then performed using a mobile agent that was launched in the remote site server where it executed the same assembly task spending about 3 minutes. 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 accepted by 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, but high levels of autonomy are required.

4.2.2. *Autonomy of the Mobile Agents*

The following set of experiments was conducted in order to show the level of autonomy of the mobile agents that is provided by the execution monitoring and error recovery capabilities.

EXPERIMENT 3. In this case an agent (mission agent) was sent to the remote site server with the abstract plan illustrated in Figure 8(a), i.e., just one high-level action. The refined executable plan is shown in Figure 8(b). The initial conditions, which coincide with the pre-conditions of the action *a1*, and the action consequence (the effects) of the experiment, are shown in Figure 9.

During the execution of this simple plan several errors were intentionally introduced. The agent recovered from the errors in all situations where a recovery plan was theoretically possible. For example, when after action *mountA(f)* an error condition was introduced telling the agent that the first part was not mounted, the agent

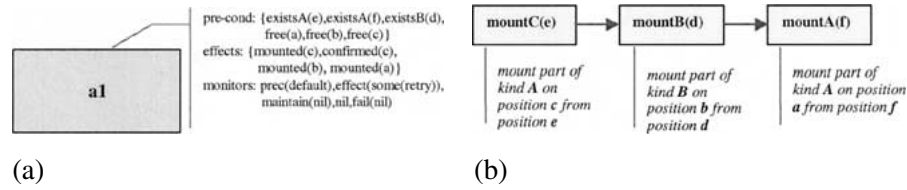


Figure 8. Abstract plan and its refinement for experiment 3.

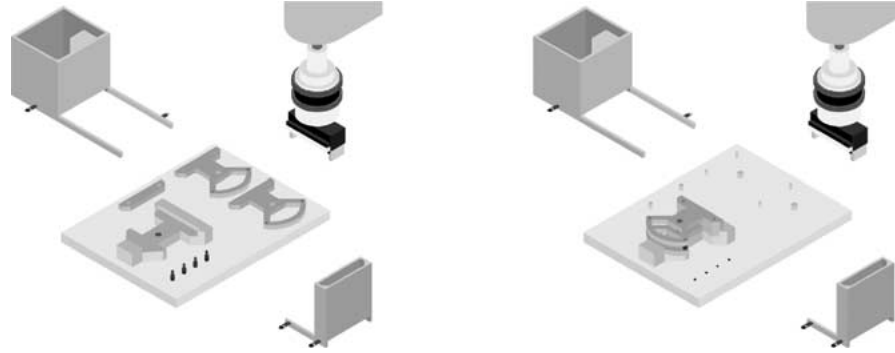


Figure 9. Initial and final conditions of the experiment 3.

fired the effect monitor of action *a1*, which indicates recovery strategy *retry*. As a consequence, the agent tried to re-execute the same executable plan, but then the pre-conditions of *a1* were not satisfied, what caused the firing of its pre-condition monitor. Since the default strategy for pre-condition monitors is *repair*, the agent generated and executed a repair plan that dismounted all parts before resuming the initial plan. Several other errors were provoked with this abstract plan and other similar plans with variations in the monitors. The agents could recover from all recoverable errors.

EXPERIMENT 4. Since this is a simple example domain, the agents from the previous experiments were able to recover from all recoverable errors, although sometimes with recovery plans involved too many actions. In order to show the advantages of the use of hierarchical plans another mission agent with the abstract plan depicted in Figure 10 was sent to the remote server. In this plan, action *a1* is refined in a lower level composed of the sequence *a2*, *a3* and *a4*. Action *a2* is in turn refined as a choice between actions *a5* and *a6*, and action *a4* is refined as a choice between *a7* and *a8*.

The initial conditions when the plan started are shown in Figure 11. An error was introduced which caused pre-condition *existsA(e)* of *a5* to fail and fire the corresponding pre-condition monitor, which in turn caused the propagation of the failure to the upper level and, consequently, fired the failure monitor of *a2* whose recovery strategy is *redo*. As a consequence, the second branch of the choice in *a2* was selected and the new plan composed of the sequence “*moveA(e)*, *mountC(e)*”

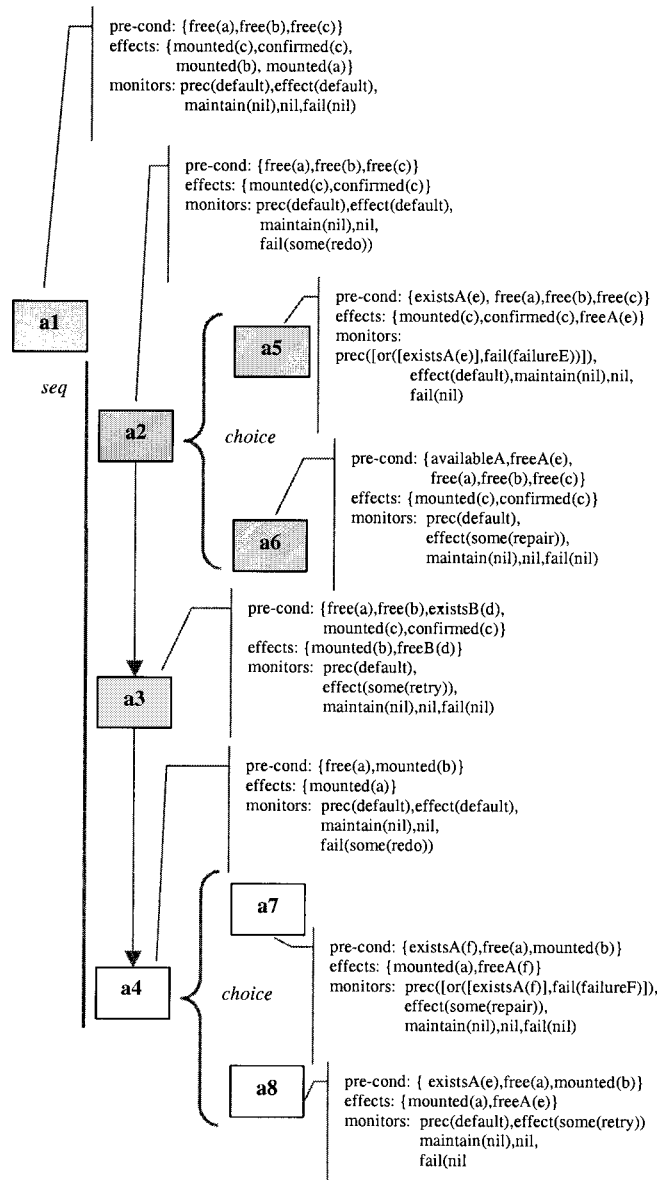


Figure 10. Abstract plan for experiment 4.

substituted the subset of the original executable plan composed only of action mountC(e) (Figure 12). However, since there was no part A available on the output of the dispenser, the pre-condition availableA of a6 failed and fired its monitor whose recovery strategy repaired the error with one single action placeA that directs the dispenser to move a new part to its output (see Figure 15). Figures 13 and 14 show these steps.

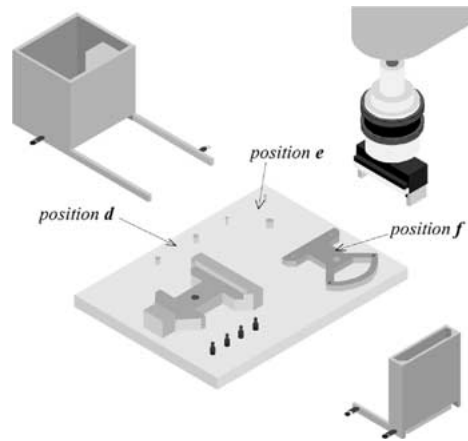


Figure 11. Initial conditions of the experiment 4.

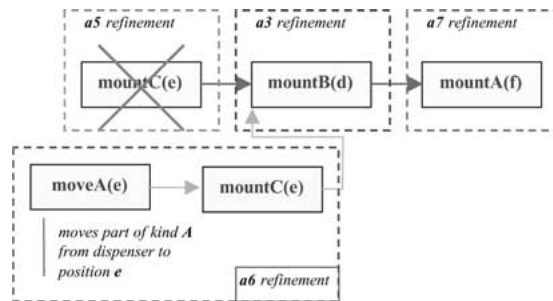


Figure 12. New executable plan resulting from failure of action a5 in the plan.

Later on in the same experiment, another error occurred because a part of type B was not available on position d, causing the pre-condition monitor of a3 to fire and repairing the plan with the sequence “placeB, moveB(d)” shown in Figure 15. This task is depicted in Figures 16 and 17.

The last action, a7, finishes the assembly task of the product.

The agent took about 5 minutes to complete the assembly task, against the 3 minutes that it would take if no errors had occurred. Again, the agent was capable of recovering from errors whenever a recovery strategy existed. On the other hand, the recovery plans tended to be smaller and more opportune than when no hierarchical decomposition was used.

EXPERIMENT 5. In Section 3 the characteristics of adapt and goal refinements were briefly introduced. With the adapt refinement the pre-conditions of the abstract action are taken as the initial state of the planner. The goal refinement causes the current real state of the world to be taken as the initial state of the planner. This experiment illustrates the two kinds of refinement. Let us consider the initial conditions depicted in Figure 18.

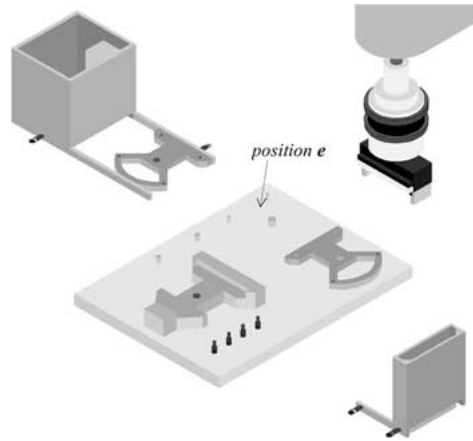


Figure 13. a6 action repair.

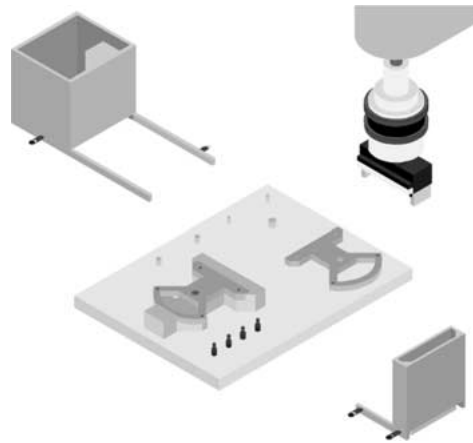


Figure 14. a6 action task.

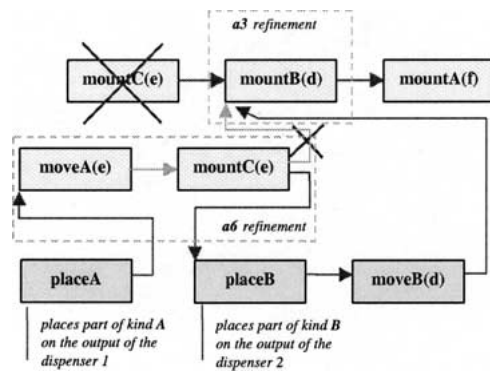


Figure 15. Repairing pre-condition existsB(d) of a3.

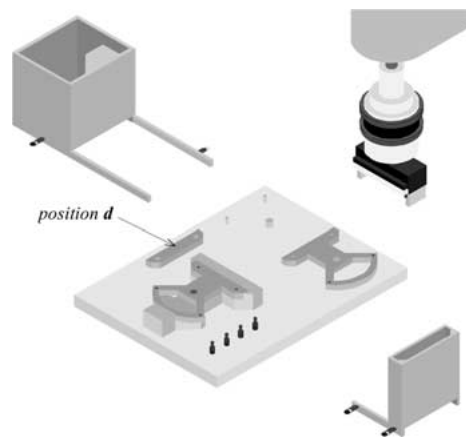


Figure 16. a3 action repair.

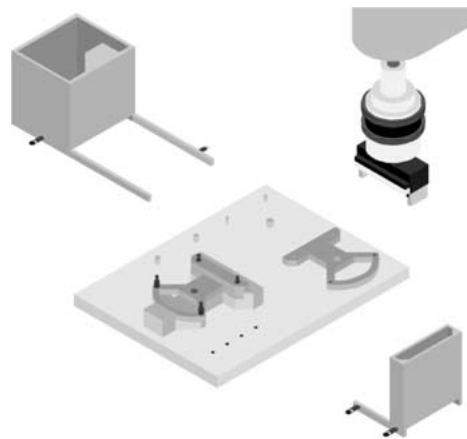


Figure 17. a3 action task.

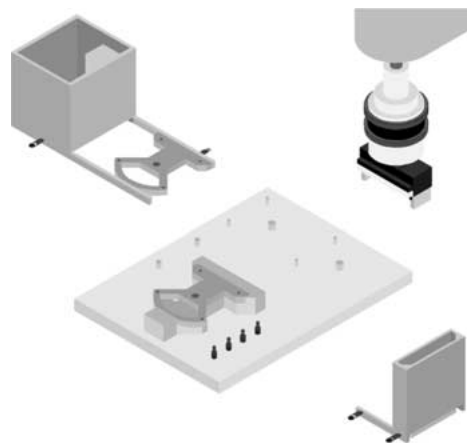


Figure 18. Initial conditions for experiment 5.

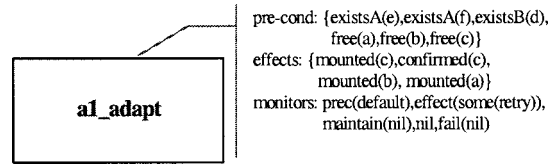


Figure 19. Plan with adapt refinement.

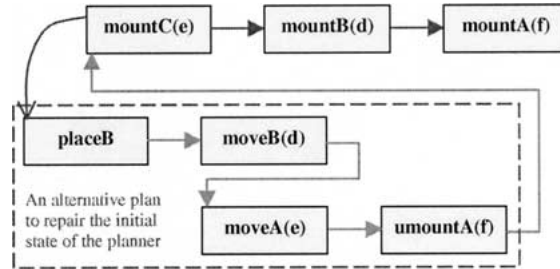


Figure 20. Refinement of the plan of Figure 19 and its alternative repair plan.

First, a plan with refinement of kind *adapt* was built as shown in Figure 19. Its refinement is shown in Figure 20. Because the initial conditions were not the pre-conditions of the plan (Figure 9), an error occurred causing the pre-condition monitor of action *a1_adapt* to fire and an alternative plan to be created, repairing the initial state of the planner (see Figure 20).

When a plan is built with a refinement of type *goal* (see Figure 21), it is not necessary to generate any alternative plan for recovering the initial conditions, because in this case the agent considers the current real state of the world and its plan is refined in one executable plan, as is shown in Figure 22.

5. Application to Virtual Laboratories

Virtual Laboratories (VL) are attracting growing interest due to their potential applications in areas such as education, research, medicine, etc., in order to operate tools and equipment in remote locations or hazardous environments, among others. A Virtual Laboratory provides a virtual experimental environment for scientists and engineers to properly perform their experiments, enabling a group of researchers located around the world to work together, sharing resources (equipments, tools, etc.) and results. A typical VL involves scientific equipments connected to a network, large-scale simulations, visualization, data reduction and data summarization capabilities, application-specific databases, collaboration tools (e.g., teleconferencing, federated data exchange, chat, shared electronic-whiteboard, notepad, etc.), application-dependent software tools and interfaces, safe communications, and large network bandwidth.

Collaborative remote supervision in manufacturing shares many requirements and challenges with the VL environment, among which the following can be mentioned:

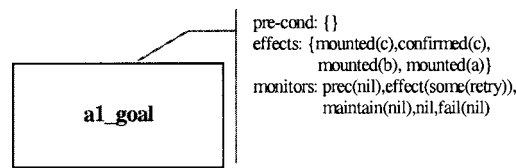


Figure 21. Plan with goal refinement.

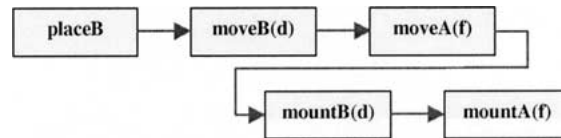


Figure 22. Refinement of the plan of Figure 21.

- Operation of remote instruments/equipment.
- Remote diagnosis.
- Team discussion, virtual meetings.
- Sharing data at highly interpreted level.
- Tele-mentoring (e.g., explaining a new process).
- What-if analysis/simulation.
- Exclusive and shared accesses (to data, tools, and devices).
- Access rights definition (to data, tools, and devices).
- Visual checks and video link (conferencing).
- Support for un-experienced users (the supervision partner does not necessarily know the details of the remote environment).
- Many adjustments previously done manually must be automated (if not assisted by local operators in the physical equipment place).

Therefore, the concept of Virtual Laboratory has to be supported by the following main requirement classes:

- *Remote operation*: in order to have access and manipulate tools and equipment located at a remote workshop;
- *Information management*: to store information and data generated by the experiments realized;
- *Simulation*: to visualize and reproduce the actions on the remote workshop, and
- *Collaborative tools*: in order to share and coordinate the experiments among different partners around the world.

This concept is illustrated in Figure 23.

Most of the works addressing the VL so far focus only on a limited part of the mentioned requirements. For instance, some developments are only focused on the simulation aspects, others on the interaction with equipments, or on some form of cooperation among researchers. There is a need for the design of general architectures that can guide the developments and interoperability among all mentioned requirements.



Figure 23. Components of a Virtual Laboratory.

One proposal in this direction is being developed at the University of Amsterdam [1]. This VL architecture is based on multiple functionality layers, so that the application/domain-specific aspects of the scientific and engineering environments can be separated from the generic VL functionality and necessary computing aspects. The generic VL functionality and computing aspects considered in this work serve a broad range of domains, by addressing issues related to distributed computing, advanced networking, archiving and basic information management, communication, cooperation/collaboration, simulation, and visualization. Furthermore, to provide a flexible and open environment for scientists and engineers using the VL, these generic functionalities are developed within a VL-abstract-machine. Therefore, the abstract machine provides a set of functions, and users can define their experiments (through a graphic interface in a workflow-like diagram) as a process constituting a number of available functions. Certain application-specific and domain-specific facilities are also defined on top of these generic functionalities. However this work is not focused on the operation and supervision of devices.

The mobile agents work described in this paper, although focused to *Remote Operation* aspects, can be considered a valuable contribute to a general VL framework and complementary to the work of the University of Amsterdam. It shall be noted that a limited *Simulation* component, providing a *feedback simulation* window, as shown in Figure 7, is also developed.

6. Discussion of Results

The conducted experiments have shown that adaptive mobile agents are a very powerful tool to implement applications where high levels of flexibility and independence on the characteristics of the communication channels are desired.

One obvious impact of the use of mobile agents is that remote tasks based on the same level of functionality implemented on the remote place can take much

less time to completion with mobile agents than with the traditional client-server approach.

The adaptive nature of the agents allows them to handle the heterogeneity of the remote places in a manner not found in other paradigms for remote operation. Of course, adaptation of the abstract plans requires that for every application domain an agreement is made on a standard taxonomy for the definition of the capabilities of the remote places.

It is also possible that some agents are unable to obtain a refined plan with the available capabilities of the remote place where they are running. In these cases the execution of the single action of the first level is terminated with a failure code that can be taken into account by the corresponding recovery strategy.

The results of the conducted experiments also show that the proposed solution allows the definition of agents with high levels of autonomy in what concerns the accomplishment of their tasks. Of course, there are limits for the autonomy of the agents which are dictated by the abstract plans they carry, by the richness of the sensorial information available in the remote places and by the fact that there may exist errors from which it is impossible to recover. Nevertheless, what the results of the conducted experiments show is that even with abstract plans composed only of one abstract level (one single action) with all monitors set equal to their default values, the agents were capable to recover from all errors for which there were recovery strategies.

One important question is how adequate is the proposed solution for real-time control. Of course, planning is well known to be a slow process that affects the overall performance of the agents. Therefore, the proposed solution is not appropriate for low-level control of critical response-time devices. However, for many applications the specification of the tasks to be performed is made at high-levels of abstraction where response-time is not as critical. If only adapt refinement strategies are indicated, it is guaranteed that the execution of the plan only begins after all the refined plan has been obtained. Thus plan complexity only affects the error recovery process which, in general, occurs seldom and is very localized and therefore has little impact on the performance of the agents. On the other hand, the use of hierarchical plans contributes to the reduction of plan complexity being thus a powerful tool for the implementation of agents with reasonable response-times.

7. Conclusions

In this paper a solution for remote operation based on adaptive mobile agents was presented. This solution allows a conciliation between the flexibility achieved with remote operation based on low level commands implemented in the remote site and the independence on the characteristics of the communication channels achieved by remote operation based on high level commands implemented in the remote site. In order to cope with the high levels of heterogeneity and autonomous evolution expected in the remote places, the agents carry hierarchical abstract plans

not completely specified. The agents obtain an executable plan by refining these abstract plans according to the capabilities they find in each site. Experiments conducted with a real robot showed that the proposed solution is quite independent of characteristics such as time-delays and availability of the communication channels. Concerning autonomy, these experiments also showed that the agents could recover from errors whenever a strategy was available.

Although the considered domain is very simple, the number of possible situations is large, being difficult to adopt deterministic solutions, especially if we consider that the same agent can visit several places with different execution environments. In fact, recovering from many of the errors introduced in the experiments would require human intervention if a traditional approach would be adopted. Therefore it is the opinion of the authors that the proposed solution is very effective in supporting remote operation over the Internet and long distance scenarios such as the spatial exploration domain.

An important application scenario for this approach is the development of infrastructures for virtual laboratories. However, for this case there are a number of open challenges requiring further research, namely, in terms of functionalities to support cooperation and experiment management.

Acknowledgements

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

References

1. Afsarmanesh, H., Benabdelkader, A., Kaletas, E., Garita, C., and Hertzberger, L. O.: Towards a multi-layer architecture for scientific virtual laboratories, in: *Lecture Notes in Computer Science* 1823, Springer, New York, 2000, pp. 163–176.
2. Cabri, G. L. and Zambonelli, F. L.: How to coordinate Internet applications based on mobile agents, in: *Proc. of WETICE'98 – Workshop on Coordination Architectures for Distributed Web Applications*, 1998.
3. Camarinha-Matos, L. M. and Vieira, W.: Adaptive mobile agents for telerobotics and telesupervision, in: *Proc. of INES'98 – 2nd IEEE Internat. Conf. on Intelligent Engineering Systems*, Vienna, Austria, September 1998, pp. 79–84.
4. Camarinha-Matos, L. M. and Vieira, W.: Intelligent mobile agents in elderly care, *Robotics and Autonomous Systems* **27**(1/2) (1999), 59–75.
5. Camarinha-Matos, L. M. and Vieira, W.: MAAPL: A language for adaptive mobile agents with execution monitoring, in: *Proc. of INES'99 – 3rd IEEE Internat. Conf. on Intelligent Engineering Systems*, Poprad, High Tatras, Stará Lesná, Slovakia, 1–3 November 1999, pp. 171–176.
6. Fikes, R. and Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving, in: J. Allen, J. Hendler and A. Tate (eds), *Readings in Planning*, Morgan Kaufmann, Los Altos, CA, 1999, pp. 88–97.
7. Fuggetta, A., Pico, G. P., and Vigna, G.: Understanding code mobility, *IEEE Trans. Software Engrg.* **24**(5) (1998), 346–361.

8. Kotz, D. and Gray, R. S.: Mobile agents and the future of the Internet, *ACM Operating Systems Rev.* **33**(3) (1999), 7–13.
9. Reece, G. and Tate, A.: Synthesizing protection monitors from causal structure, in: *Proc. of the 2nd Internat. Conf. on Planning Systems*, AAAI Press, 1994.
10. Vieira, W.: Adaptive mobile agents for remote operation (in Portuguese), PhD Thesis, New University of Lisbon, 2000.
11. Vieira, W. and Camarinha-Matos, L. M.: Execution monitoring in adaptive mobile agents, in: M. Klusch, O. Shehory and G. Weiss (eds), *Cooperative Information Agents III*, Lecture Notes in Artificial Intelligence 1652, Springer, Berlin, 1999, pp. 220–231.
12. Weld, D.: Recent advances in AI planning, *AI Magazine* (1999).
13. White, J. E.: Mobile agents, in: J. M. Bradshaw (ed.), *Software Agents*, MIT Press, Cambridge, MA, 1997, pp. 437–472.
14. Wooldridge, M.: Intelligent agents, in: G. Weiss (ed.), *Multiagent Systems*, MIT Press, Cambridge, MA, 1999.