Embedded Control of Autonomous Robots using Procedural Reasoning *

François Félix Ingrand Raja Chatila Rachid Alami
Frédéric Robert
LAAS - CNRS
7, Avenue du Colonel Roche 31077 Toulouse Cedex - France
e-mail: {felix, raja, rachid, fr}@laas.fr

Abstract

We describe an architecture for action planning and execution control for mobile robots based on the use of a planner/supervisor pair that associates deliberation and reactivity. The supervision component is based on the Procedural Reasoning System (PRS) that controls the execution of scripts and plans in dynamic environments. The robot's actions are executed by modules embedding the necessary processings for action and perception.

1 Introduction

An autonomous robot should be able to anticipate its future actions to achieve its tasks efficiently. In other words, it should be able to plan for its actions. On the other hand, it should be able to cope with the evolution of the environment and of its own state, i.e., to be reactive. This reactivity should be a feature at all levels of decision, not only at immediate reflex reactions.

We propose an architecture (Figure 1) that answers the requirements for robot autonomy composed of a *functional level* embedding the robot sensing and acting capacities, including closed-loop processes, and a *decisional level* based on a layered plan-based framework that integrates interaction between deliberation and action [5].

We shall focus on this aspect of embedded control and supervision, and describe the use of PRS (Procedural Reasoning System, first developed at SRI [6]) as a tool for implementing the interaction between deliberation and reaction. PRS, and its implementation C-PRS that we use, provides tools and mechanisms to represent and execute plans, scripts and procedures, i.e., conditional sequences of actions which can be run to achieve given goals or to react to particular situations.

2 Deliberation and Action in the Robot System

In the robot architecture, the *Supervisor* interacts with the other layers and with the refinement planner. The other layers are viewed as a set of processes which exchange signals with the supervisor. These processes correspond to the actions of the agent as well as events associated with environment changes independent from robot actions.

The planner is given a description of the state of the world and a goal; it produces a plan (e.g., the sequence of motions and perception actions to take). One criterion that should be considered when speaking about planning is the "quality" of the produced plan which is related to the cost effectiveness of achievement of a given task or objective (time, energy,...), and to the robustness of the plan, i.e., its ability to cope with non-nominal situations. This last aspect is one of the motivations of our approach: besides providing a plan, the planner should also provide a set of execution "modalities" expressed in terms of:

- constraints or directions to be used for execution. These directions may be considered as meta-knowledge for the supervision of execution.
- description of situations to monitor and the appropriate reactions to their occurrence in order to prepare a more effective robot behavior to some possible events; such reactions are immediate reflexes, "local" correcting actions (without questioning the plan), or requests for replanning.

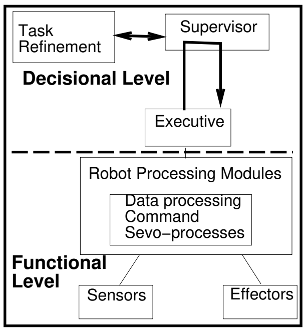

Figure 1: Architecture for an Autonomous Robot

The activity of the supervisor consists in monitoring the plan execution by performing situation detection and assessment and by taking appropriate decisions in real time, i.e., within time bounds compatible with the rate of the signals produced by the processes and their possible consequences.

The responsibility of "closing the loop" at the level of plan execution control is entirely devoted to the supervisor. In order to achieve it, the supervisor makes use only of deliberation algorithms which are guaranteed to be time-bounded and compatible with the dynamics of the controlled system.

*This paper will appear in the proceedings of 1995 ICAR, San Feliu de Guixols, Spain.

This execution control is done through the use of the plan and its execution modalities, as well a set of *situation-driven procedures* embedded in the supervisor and independent of the plan. These procedures are predefined at design phase. They can take into account the current goal and plan when they are executed, by recognizing specific goal or plan patterns.

The execution processes are represented by *finite state automata* (FSA). In the FSA we use, the set of allowed external signals correspond to all the actions that can be taken by the supervisor. Similarly, the set of possible internal signals correspond to all environment changes that could be perceived by the supervisor. The execution processes are embedded in robot modules controlled by an "Executive" implemented as a compiled 0^+ rule based-system, that produce a bounded-depth decision tree.

It is important to note that the supervisor is not just an interpreter that would execute a "reactive plan" composed of the plan and modalities produced by the planner. Indeed, the supervisor actually makes evaluations and takes decisions on the way the actions should be executed. Furthermore, the supervisor may decide that a replanning of a task is necessary, and in this sense it also controls the planner considered as a resource, and it may ask for a new mission plan or decision from the control station.

The presented architecture, including PRS, is currently used in two different projects of autonomous robots that exhibit various features of the system.

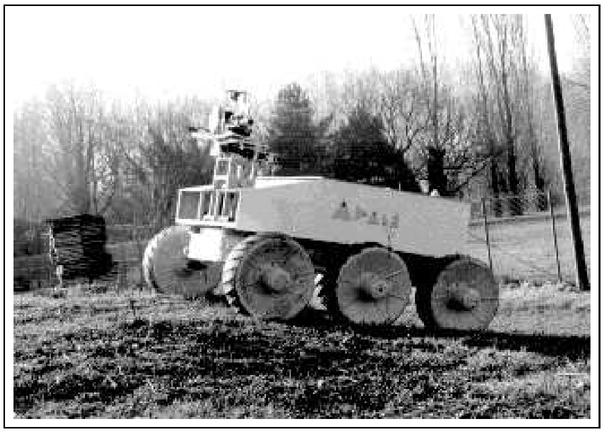

Figure 2: *The mobile robot Adam.*

Eden The Eden experiment (using the Adam robot, Figure 2)) aims at demonstrating a fully autonomous navigation in a natural environment, including perception, environment modeling, robot localization, and motion planning and execution on flat or uneven terrain. Application of such developments are for example planetary exploration, and more generally intervention robots. i.e., robots sent to accomplish tasks in ill-known environments (planets, undersea, hazardous areas).

The robot receives missions from a Ground station defined at a *task-level* such as to pick up a rock, or to reach a given place that it has to interpret into plans to execute according to the actual context that evolves during execution. The canonical task that we experimented in Eden is the navigation task "GO-TO (Landmark)" in an initially unknown environment gradually discovered by the robot. The landmark is an object only known by its model (and not necessarily its location) which can be recognized and localized on a video or range image. The decisions taken by the robot are related to the navigation modes to be adapted to the nature of the terrain, and to the perception of new data for updating robot position with respect to landmarks acquisition of unknown parts.

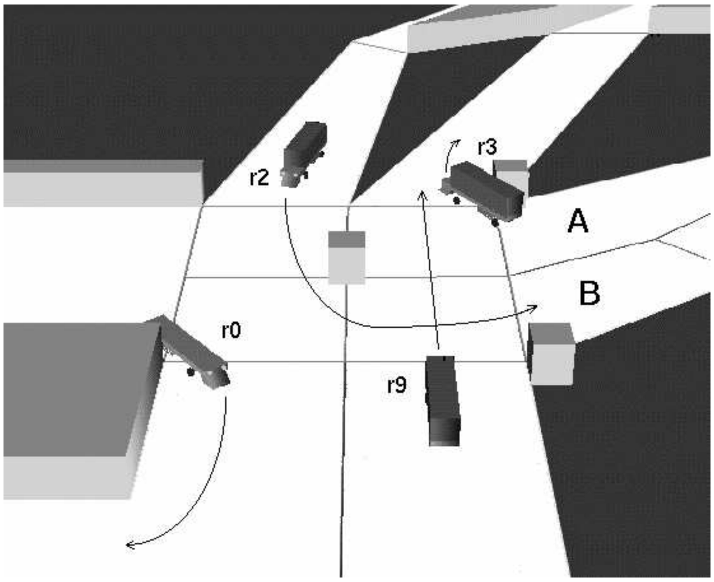

Figure 3: A typical situation in MARTHA

Martha The second project concerns a multi-robot application developed in in the framework of the MARTHA project[1] which deals with the control of a large fleet of autonomous mobile robots for the transportation of containers in harbors, airports and railway environments. The number of robots in the fleet is in the order of 50

In such context, the dynamics of the environment, the impossibility to correctly estimate the duration of actions (the robots may be slowed down due to obstacle avoidance, and delays in load and un-load operations, etc..) prevent a central system from elaborating efficient and reliable detailed robot plans.

Thus, the robots are endowed with all the necessary ingredients for planning and executing navigation and load handling missions expressed at a very high level as well as for multi-robot cooperation.

Multi-robot cooperation is based on a generic paradigm called *Plan-Merging Paradigm* [2], where robots incrementally merge their plans into a set of already coordinated plans. This is done through exchange of information about their current state and their future actions. Figure 3 shows a typical situation in Martha, where a number of robot must coordinate their plan to traverse a crossing.

3 Procedural Reasoning for Supervision

Procedural reasoning is a suitable framework for implementing the supervisor part in the robot architecture. Before discussing how it is used we first present a brief description of its main features.

[1]MARTHA: European ESPRIT Project No 6668. "Multiple Autonomous Robots for Transport and Handling Applications"

3.1 The Procedural Reasoning System

PRS is composed of a set of tools and methods to represent and execute plans and procedures. These plans or procedures are conditional sequences of actions and goals which can be run or posted to achieve given goals or to react to particular situations. Procedural reasoning differs from other commonly used knowledge representations (rules, frames, ...) as it preserves the control information (i.e. the sequence of actions and tests) embedded in procedures or plans.

A complete description of PRS is given in previous papers [10]. Nevertheless, we find it necessary to provide a brief description of its main components:

a database which contains facts representing the system view of the world and which is constantly and automatically updated as new events appear,

a library of procedures (or scripts), each describing a particular sequence of actions and tests that may be performed to achieve given goals or to react to certain situations,

an intention (or task) graph which is a dynamic set of intentions/tasks currently executing (Figure 7 shows an example of an intention graph snapshot (from a multi robot experiment performed at LAAS)). Intentions (or tasks) are dynamic structures which execute the "intended procedures", they keep track of the state of execution of these intended procedure, and of the state of their posted subgoals.

There exist various implementations of PRS: SRI PRS [10], ADS PRS, UM-PRS [12]. The one we use, and present in this paper is called C-PRS [1] and is an implementation of PRS in C, under Unix.

3.1.1 KAs, Scripts and Procedures

Knowledge about how to accomplish given goals or to react to certain situations is represented in PRS by declarative procedures historically called *Knowledge Areas* (KAs). (See Figure 4 and 5). Each KA consists of a *body*, which describes the steps of the procedure/plan[2], an *invocation condition*, which specifies the goal the KA may fulfill or the events to which it reacts, and a *context* describing under which situations the KA is applicable. Together, the invocation condition and body of a KA express a declarative fact about the results and utility of performing certain sequences of actions under certain conditions [6]. Other piece of information are stored in KA such as facts to conclude or retract upon successful execution or properties which hold user-defined property/value pairs (See Figure 5).

In PRS, *goals* are descriptions of a desired state associated to a behavior to reach/test this state. For example, the goal to position `robot-1` in `dock-area-47` is written `(ACHIEVE (position robot-1 dock-area-47))`. The goal to test (without modifying the environment) if the robot `robot-1` is in `dock-area-47` is represented `(TEST (position ...))`. The goal to passively wait until the robot `robot-1` gets in `dock-area-47` is represented `(WAIT (...))`. The goal to check that the robot `robot-1` stays in `dock-area-47` while performing other actions is represented `(PRESERVE (...))`. Similarly, the goal to maintain the robot `robot-1` in `dock-area-47` is represented `(MAINTAIN (...))`. For example, the goal to follow another robot while maintaining a safe distance in between: `(& (ACHIEVE (follow robot-2)) (MAINTAIN (safe-distance @@myself robot-2)))`.

The KA presented on Figure 4 is used in the Eden experiment to get the robot moving. After some initialization, one can note that this KA will loop if an improper reply is received from the Laser Range Finder (which in this case means that an obstacle has been detected). Otherwise, it will exit through the `N9 T` branch.

3.1.2 Meta-level KAs

The set of KAs in a PRS application system not only consists of procedural knowledge about a specific domain (See Figure 4), but also includes *meta-level* KAs (See Figure 5) – that is, KAs able to manipulate applicable KAs, goals, and intentions of PRS itself. The use of meta-level KAs ranges from methods for choosing among multiple applicable KAs, to insure mutual exclusion on critical resources, or to compute the amount of additional reasoning that can be undertaken, given the real-time constraints of the problem domain. To achieve such objectives, these meta-level KAs make use of information about KAs, goals, facts that is contained in the system database or in the *properties* slot of the KA. For example, the meta KA on Figure 5 will insure that any fact invoked procedure because of an external event (`FACT-INVOKED-KAS`), will be intended in the proper place in the intention graph (depending on some properties of the KA), while the goal invoked KAs will be chosen randomly (after all, their goal will remain unfulfilled and will be automatically reposted by the kernel)[3].

3.1.3 The Interpreter

The PRS kernel interacts with its environment both through its database, which acquires new beliefs in response to changes in the environment, and through the actions it performs as it carries out its intentions.

As shown in Figure 6, an interpreter manipulates these components. It receives new events (both from outside and from asserted facts) and internal goals (1), selects an appropriate KA based on these new events, goals, and system beliefs (2), places the selected KA on the intention graph (3), chooses a current intention/task among the roots of the graph (4) and finally executes *one step* of the active KA in the selected intention (5). This can result in a primitive action (6), or the establishment of a new goal (7).

An important part of the main loop is the one which finds applicable KAs and selects those which will be intended (2). Basically, this part is composed of one meta-level reasoning loop inside the main loop. The purpose of this inner loop is to determine the successive sets of applicable KAs, in the light of the concluded beliefs on the previous set of applicable KAs (See Section 3.4.1).

3.2 Intention/Task Graph

The Intention/Task Graph plays an important role in PRS. Figure 7 shows an example of an intention graph snapshot, from a multi robots experiment performed at LAAS. It holds all the intentions/tasks which have been intended (in this particular example, one can see the various tasks performed by the system: Mission supervision, Mission

[2] Some KAs, called action KAs, just have an external function call as a body.

[3] This type of KA is usually required in a monitoring and control application where external events are considered more urgent than internal goals.

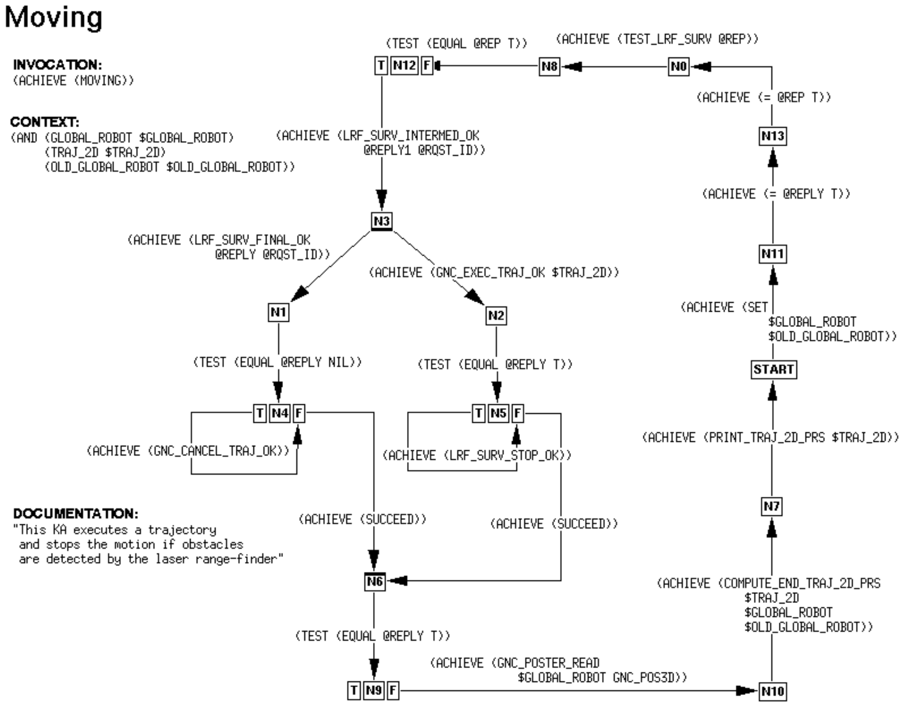

Figure 4: A KA with multiple threads

refinement, Coordination, and so on, as well as the meta level intention currently intending newly applicable KAs). Indeed, when a KA is applicable, the system (i.e., the main loop or some meta KA execution) may decide to intend it (i.e., to propose it for execution). Depending on the origin of the KA instance (if it was invoked by a goal or by an event), it can either result in a new intention or it can be put on the top of the execution tree of the intention it was posted from.

So if a KA is applicable because of an event or because of a goal posted by the user (thus an external goal), then the resulting KA instance is intended in a new intention. But if a KA is applicable because of a goal posted by an already executing KA (which has been intended previously), then this KA instance will be intended in the same intention.

This distinction makes sense as KAs pursuing a goal posted by the system would be intended in the same intention/task than the one their goal originated from. On the contrary, KAs applicable because of changes in the world, i.e. events, are not explicitly connected to any previous activity.

3.3 Implementation of a Supervisor using C-PRS

There are a number of reasons why the PRS approach appears to be well suited for the implementation of a supervisor which satisfies the requirements of the above mentioned architecture. We shall now examine them. Some of these reasons relate to the original capabilities of the PRS and some relate to new features implemented in C-PRS, the specific version we use.

3.3.1 Partial Plan/Script Representation

In PRS, each KA is self-contained, as it describes in which condition it is applicable and the goals it achieves. It usually contains in its "body" tests which condition the proper posting of its subgoals while leaving to the interpreter (and the meta-level KAs) the choice of the adequate procedure to try to satisfy each posted subgoal.

This is particularly well adapted to context based task refinement and to a large class of robot tasks which can be viewed as incremental. The same task corresponding to the achievement of a given goal has to be pursued for a given period of time while its conditions change due to its own execution state or to changes in the environment state or the robot state. A typical task of this type is navigation is a partially known and/or dynamic environment.

For example, in the experiments we have performed on mobile robot at LAAS, a number of tests and actions must be performed before the robot begins to plan its motion. Nevertheless, the choice of the motion planner used is left to the interpreter and possibly to meta-level KAs which will decide which method is the best in the current situation.

3.3.2 Event and Goal Driven Behavior

KAs and procedures can be triggered upon occurrence of events or posting of goals. For example when a procedure is executed, it will post subgoals to be satisfied. These subgoals will be either immediately satisfied if their statement is already present in the database, or will lead to the execution of other procedures the invocation part of which unifies with the posted subgoal. In fact, the notion of goal in PRS is rather strong as it really represents an objective the interpreter will try to satisfy by any means, i.e. by trying one after another the procedures which unify with it (the applicablility of the procedure is reevaluated after each KA failure). As a consequence, a goal is considered as failed after *all* KAs for each valid unification binding have been already tried, and have failed.

The notion of event differs from the notion of goal

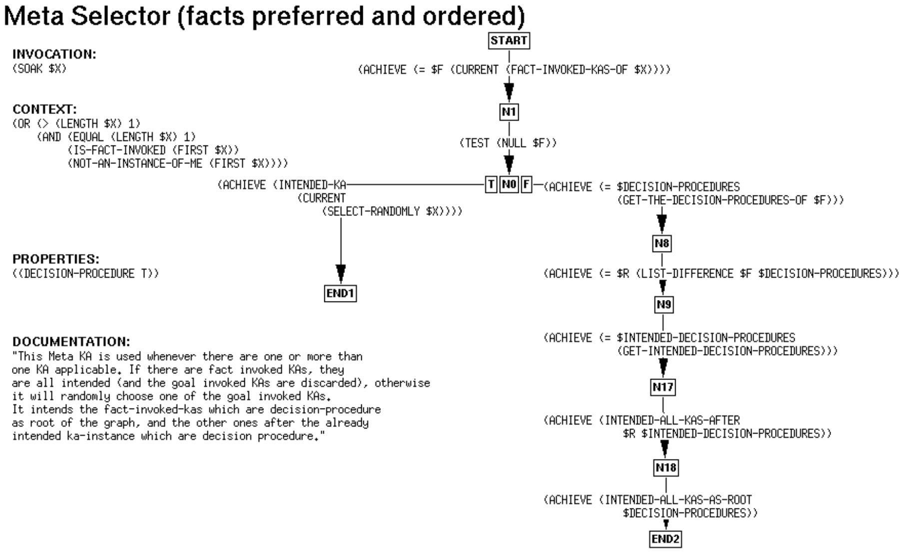

Figure 5: A Simple Meta Level KA

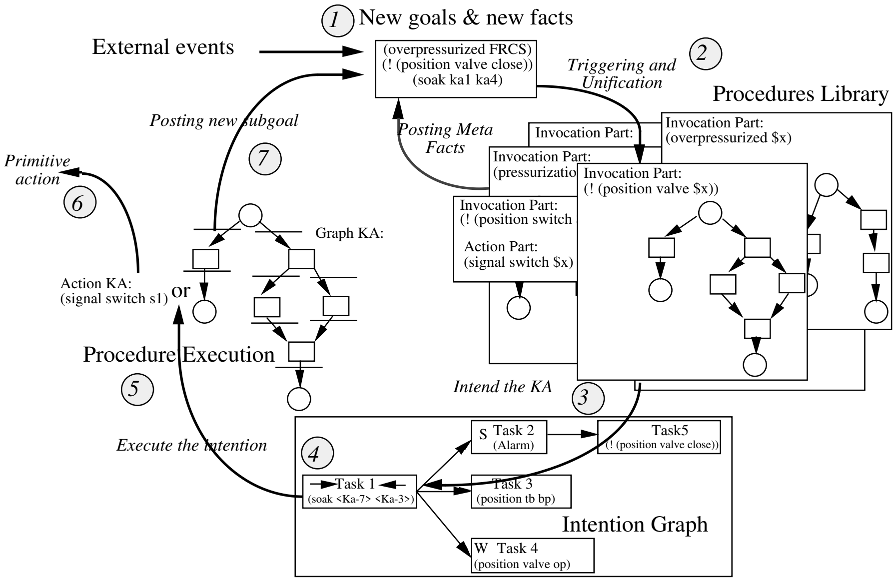

Figure 6: PRS Interpreter

for two reasons. First, it may render some procedures applicable, but these procedures do not pursue an explicit goal. They merely reply to events and produce subgoals to be achieved without any visible objective, and the overall success or failure of these event-driven procedures is not analyzed by any means. Second, events represent new pieces of information which are usually stored in the database for further reference and to update the state of the world as seen by the system.

These two types of behavior, event and goal driven, actually correspond to two types of activities. Event-driven procedures are generally used to monitor conditions which require some attention from the system, while the goal-driven procedures are used to pursue a particular objective in a given context.

3.4 Supervision and Monitoring

One can monitor a condition using procedures with this particular condition in their invocation part. However, there are a number of ways to implement more sophisticated supervision and monitoring.

The KA in Figure 4 illustrates one of these mechanisms. Out from node `N3`, two threads are started, one to execute a trajectory, another one to set a monitoring. If the trajectory executes properly, it then stops the monitoring task which will return without modifying the `@REPLY` variable. Otherwise, if the monitoring detects an obstacle, it returns and the `@REPLY` variable is set to nil, which leads this thread to cancel the trajectory execution.

Other types of monitoring and supervision can be implemented using the `WAIT`, `PRESERVE` and `MAINTAIN` operators.

3.4.1 Reasoning on Robot Capabilities

The meta level reasoning available under PRS provides a mechanism to control the PRS main loop. Currently, meta level reasoning is mainly used in the

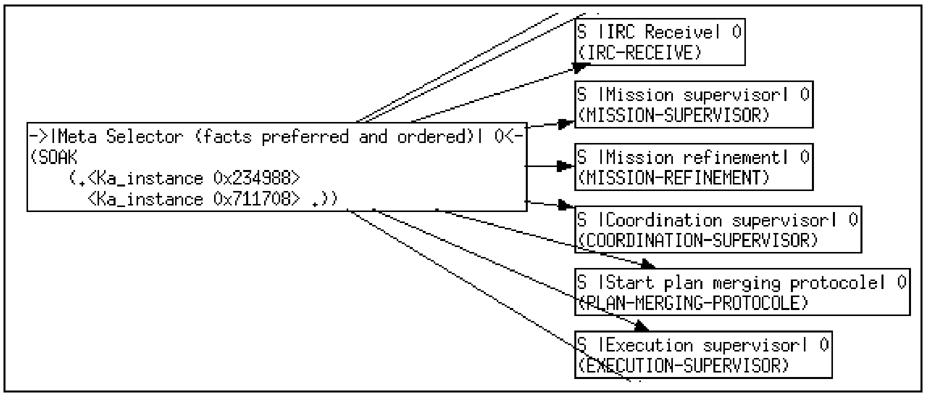

Figure 7: Partial Intention Graph Snapshot

KA selection part of the PRS main loop[4].

As explained before, new events and new goals lead to some KAs becoming applicable. Before intending any of these KAs, this very information, i.e the fact that some KAs are applicable, is itself asserted in the system. As any other fact/event, it may lead to some KAs (called meta KAs) becoming applicable again. This reflexive cycle ends when no "new" applicable KAs are found in a loop. This means that there exist no more criteria (i.e., an applicable meta KA) to select among the applicable KAs. The interpreter then chooses randomly a KA to intend in the set of the KAs applicable in the previous cycle. Most likely, this chosen meta KA will then take care of intending all or part of the applicable KAs which have lead to its own applicability. This mechanism which may seem awkward and prone to applicable KAs being "forgotten" is in fact the least constraining to leave the full control to user-defined meta KAs. Figure 5 presents an example of a meta KA.

In applications such as the ones presented above, meta level reasoning can be used for a number of reasons, e.g., to ensure mutual exclusion of the execution of incompatible procedures, or to implement a preference on the method used to achieve a particular goal when multiple alternatives are given, or to implement some event or procedure based priority mechanism.

3.4.2 Time bounded reaction

As discussed above, one mandatory feature of the supervisor is its ability to react with a guaranteed time time bound compatible with the dynamics of the controlled system.

The algorithms and the main loop used in C-PRS are such that, under some reasonable assumptions, the C-PRS main loop can guarantee an upper bound on reaction time (see [9]). This upper bound is a function of the longest action of the system and of the maximum frequency of event arrival. From this, the user can derive or implement other complex and advanced temporal properties, such as priority mechanisms, deadlines, and so on.

For example, using meta level KAs, it is easy to implement a mechanism which can guarantee that a KA with a particular property will be intended as root of the intention graph (therefore executed before any other KA).

3.4.3 Other useful C-PRS features

C-PRS provides a set of features which allow its effective use in autonomous robot applications. Besides complementary constructions which facilitate programming (e.g. , Text KAs, Multi-threading, Elaborated programming environment), C-PRS provides mechanisms to allow its integration with other systems.

A communication library (called mp-prs) has been developed which provides simple but efficient communications over Internet sockets between C-PRS processes and other processes. It is also possible to link user C functions to the C-PRS kernel which will be invoked through an action KA or when evaluating a PRS function or predicate.

Another key aspect is the availability of C-PRS on Unix workstations but also 680X0 or Sparc board under VxWorks. This enables an easy migration of the application from the workstation to the robot's on-board CPUs.

3.5 Future Developments

There are a number of developments which we think would improve the overall capabilities of C-PRS to handle supervision and control of mobile robots.

An important issue which remains open in the current version of C-PRS is the ability to execute a subset of the loaded KAs with a guaranteed bound on their execution time. This could be achieved using compilation techniques similar to the ones used in KHEOPS [8], a 0^+ rule based-system, that produce a bounded-depth decision tree or using situated automata such as in Rex/Gapps [11].

Another point which appears critical in the two mentioned applications is the ability to handle errors at the "procedure" level. We could implement in C-PRS some kind of error handling mechanism on procedures. Each procedure would then have a number of error handlers which trigger under specified conditions or with particular signals. These handlers could be implemented using the internal mechanisms currently used by the `PRESERVE` and the `MAINTAIN` operators.

Last, we think that the notion of activity, which corresponds to tasks under execution, although more or less present in the intention/task concept, must be further developed to be easily handled by the user [5]. The activity tree is an important representation level

[4] However, it can easily be extended to other parts of the PRS interpreter (for example to react to intention graph changes or to goal failure).

in mobile robot control as it allows to send events or signals to activities and propagate them to its children. This notion would improve the control mechanism because it represents more accurately the status of the robot execution tasks.

4 Related Work

The earliest work on procedural reasoning is the study performed by Georgeff *et al* at SRI and described in [7]. One of the major criticisms one can make to this study is that it never reached a point where a real robot ran under the control of SRI PRS. For various reasons, but mainly performances, the procedures were ran with a robot simulator. Moreover, the version of SRI PRS used at that time lacked many of the functionalities which now make an implementation such as C-PRS better suited for this type of application.

More recently, other research laboratories have found interest in using the PRS approach for mobile robot applications. In [12], the authors describe an implementation of procedural reasoning (called UM-PRS) to control an outdoor environment vehicle.

A well known architectural paradigm for robot control is NASREM [3]. However, this is not actually a system, but rather a set of guidelines and mechanisms for implementing hierarchical control systems.

A global architecture, TCA, for autonomous robot control was also proposed by R. Simmons [13]. The architecture features several properties that we find in the one presented above, and is in the same "school of thought" of integrating deliberation and reactivity, as opposed to behavior-based approaches which are rather event-driven [4]. TCA is based on a central control that handles several classes of messages exchanged with specific modules. It manipulates a task tree similar in a way to the intention graph in PRS. TCA also includes explicit temporal constraints on tasks. One important difference with the PRS approach is that the execution of a KA is questioned at each step, with respect to the content of the database. This is a powerful feature for easy and incremental programming and for adapting the execution to the context.

5 Conclusion

We presented in this paper a generic architecture for autonomous robots. The robot control system is composed of a a decisional level and a functional level based on a distinction between decision based on global and abstract representations and computations on low level numerical representations. The decisional level is composed of a planner-supervisor pair to ensure a deliberative and reactive behavior. The supervisor makes use of procedural representations for plan execution and goal refinement, and pursues goal-directed tasks while being responsive to changing patterns of events in bounded time. The use of PRS to implement the supervisory part was detailed and its critical features presented. Work on refining this architecture and improving some features in PRS is on-going to better suit control and supervision of autonomous mobile robots.

References

[1] ACS Technologies. *C-PRS Development Environment Manual*. ACS Technologies, 5, Place du Village d'Entreprises, BP 556, 31674 LABEGE Cedex, France, 1992-1994.

[2] R. Alami, F. Robert, F. F. Ingrand, and S. Suzuki. A paradigm for plan-merging and its use for multi-robot cooperation. In *IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas (USA), October 1994.

[3] J. S. Albus, H. G. McCain, and R. Lumia. NASA/NSB Standard Reference Model for Telerobot Control System Architecture (NASREM). Technical Report NSB Technical Note 1235, US DEPARTEMENT OF COMMERCE National Bureau of Standards, 1987.

[4] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, April 1986.

[5] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. Integrated planning and execution control of autonomous robot actions. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, 1992.

[6] M. P. Georgeff and A. L. Lansky. Procedural Knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74:1383–1398, 1986.

[7] M. P. Georgeff, A. L. Lansky, and M. Schoppers. Reasoning and planning in dynamic domains: an experiment with a mobile robot. Technical note 380, AI Center, SRI International, Menlo Park, California (USA), 1987.

[8] M. Ghallab and H. Philippe. A compiler for real-time Knowledge-based Systems. In *International Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, Japan, May 1988. IEEE.

[9] F. F. Ingrand and V. Coutance. Real-Time Reasoning using Procedural Reasoning. Technical Report 93-104, LAAS/CNRS, Toulouse, France, 1993.

[10] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992.

[11] L. P. Kaelbling. Goals as Parallel Program Specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 60–65, Saint Paul, Minnesota (USA), 1988.

[12] J. Lee, M. J. Huber, E. H. Durfee, and P. G. Kenny. UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications. In *Proceedings of the Conference on Intelligent Robotics in Field, Factory, Service, and Space*, pages 842–849, Houston, Texas (USA), March 1994.

[13] Reid G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), Feb. 1994.