PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots *

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

In this paper, we discuss Procedural Reasoning System (PRS) as a high level Control and Supervision language adapted to autonomous robots to represent and execute procedures, scripts and plans in dynamic environments. We discuss the main reasons why PRS is well suited for this type of application: (1) The semantics of its plan (procedure) representation, which is important for plan execution and goal refinement; (2) Its ability to construct and act on partial (rather than complete) plans; (3) Its ability to pursue goal-directed tasks while at the same time being responsive to changing patterns of events in bounded time; (4) Its facilities for managing multiple tasks in real-time; (5) Its default mechanisms for handling stringent real-time demands of its environment; and (6) Its meta-level (or reflective) reasoning capabilities. C-PRS[1] *has been used to implement an embedded control and supervision system for autonomous mobile robots in two different experimentations that we briefly present (Eden and Martha). We conclude with some suggestions to further develop C-PRS and with a short review of related work.*

1 Introduction

The study and the design of high level control architectures for autonomous robots, have always been an important research subject at LAAS [4, 8]. Part of this research is concerned with the design of tools and languages to implement components of these architectures. In recent years, we started to use the Procedural Reasoning System (PRS) to implement the control and supervision component. Other components, using other languages and tools, are used in these architecture and are linked to PRS. For example, a high level planner can be used to pass plans to PRS for execution. Similarly, to access and control the mobile robot, PRS is asynchronously connected to low level modules [5] . These other components are not discussed in this paper (although we mention how C-PRS may interface to them).

*This paper has been published in the Proceedings of the 1996 IEEE International Conference on Robotics and Automation (Minneapolis, USA).

[1] An implementation of PRS in C.

PRS implements a generic tool for representing actions and procedures and reasoning about them. It provides tools and mechanisms to represent and execute plans, scripts and procedures,[2] i.e., conditional sequences of actions which can be run to achieve given goals or to react to particular situations. C-PRS is used in the Martha and the Eden demonstrations, two large experimentations developed at LAAS. The Eden experiment 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 (see [12] for more information). Martha is an European Esprit project which objective is to study the problem of planning and controlling the actions of a fleet of autonomous mobile robots to handle container transportation in harbors, airports and railway stations (see [1] for a more complete presentation).

In these two applications, C-PRS is used as the onboard reactive decisional system. It does not plan the high level mission, nor does it engage in low level control loops. It merely executes predefined plans, making the runtime decisions specified by these plans, current policies and/or modalities. This activity is often referenced as reactive planning as it does not produce new plans from action descriptions as would a standard planner. Nevertheless it reasons about and decides, at run time, what are the best options to fulfill the current goals. These run time decisions can either be specified as explicit tests in a given plan, or as application contextual conditions of the plan or, provided that more than one plan is applicable to the same goal, as a general policy controlling the choice of which plan gets executed.

2 The Procedural Reasoning System

PRS[3] is composed of a set of tools and methods to represent and execute plans and procedures. Procedural

[2] The terms plans, scripts, procedures and KAs (for Knowledge Area) are used interchangeably in this paper. We are aware that they do not really mean the same object, but as far as PRS is concerned, the data structure holding them is the same.

[3] We shall refer to PRS as the general concept of Procedural Reasoning System and to C-PRS as the specific implementation used.

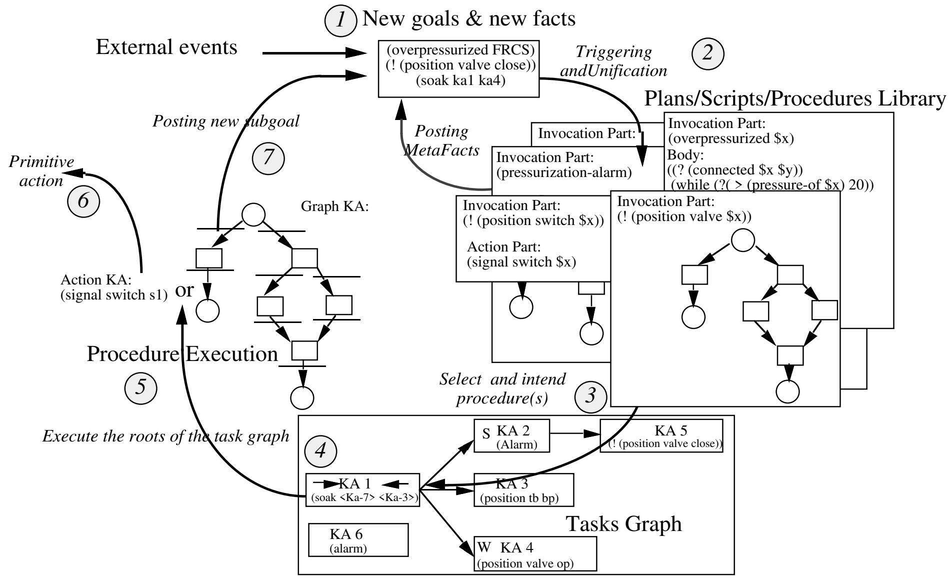

Figure 1: PRS Interpreter

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, while keeping some declarative aspects.

A description of PRS is given in previous papers [10]. Nevertheless, we find it necessary to present the architecture of a PRS kernel here (with examples in the mobile robot world). A PRS kernel is composed of three main elements:

a database which contains facts representing the system view of the world and which is constantly and automatically updated as new events appear. In typical robotic applications, the database may contain symbolic but also numerical information such as the position of the robot, the container it carries, pointers to trajectories produced by a motion planner, the currently used resources, etc. Note that the database is not just a place holder for information related to the robot and its environment. Mechanisms are provided to define evaluable predicates which, although they look like if they are in the database, call or trigger some internal C code to retrieve their value/binding.

a library of plans (or 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. The content of this plan/procedure library is definitely application dependent. In any case, keep in mind that PRS does not plan by combining actions, but by choosing among alternative plans/or execution paths in an executing plan. Therefore this library must contain all the plans/-procedures/scripts needed to perform the tasks for which the robot is intended.

a tasks graph which is a dynamic set of tasks currently executing. Tasks are dynamic structures which execute the "intended plans", they keep track of the state of execution of these intended procedure, and of the state of their posted subgoals. This task graph can be thought of as the set of processes of an operating system. For example, in a mobile robot application, the task graph could contain the tasks corresponding to various activities the robot is performing (one activity to refine its current mission, another to monitor incoming messages from a central station giving orders, another one managing the communication layer with low level functional modules, etc) (see Figure 6).

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

2.1 Plans, Scripts and Procedures

Information about how to accomplish given goals or to react to certain situations is represented in PRS by declarative plan/procedures. We illustrate the following presentation with two procedures:[4] a text procedure on

[4]Although the syntax remains Lisp like, there is no Lisp inter-

```
(defka |Long Range Displacement|
:invocation (achieve (position-robot $x $y $theta))
:context (and (test (position-robot
          @current-x @current-y @current-theta))
     (test (long-range-displacement
          $x $y $theta @current-x @current-y
          @current-theta)))
:body ((achieve (notify all-subsystems displacement))
 (wait (V (robot-status ready-for-displacement)
        (elapsed-time (time) 60)))
 (if (test (robot-status ready-for-displacement))
     (while (test (long-range-displacement
                  $x $y $theta @current-x
                  @current-y @current-theta))
       (achieve (analyze-terrain))
       (achieve (find-subgoal $x $y $theta
                             @sub-x @sub-y @sub-theta)
       (achieve (find-trajectory $x $y $theta @sub-x
                             @sub-y @sub-theta @traj)
       (& (achieve (execute-trajectory @traj))
          (maintain (battery-level 0.200000)))
       (test (position-robot @current-x @current-y
                             @current-theta)))
   (achieve (position-robot $x $y $theta))
   else
   (achieve (failed)))))
```

Figure 2: A Plan for Long Range Displacement

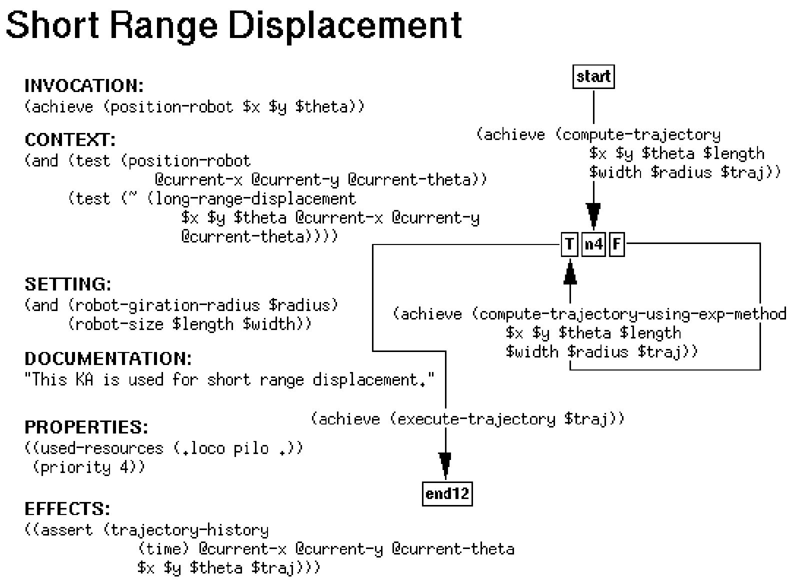

Figure 3: A Plan for Short Range Displacement

Figure 2 and a graphic procedure on Figure 3. These procedures are not withdrawn from our applications, they merely illustrate the various mechanisms provided by PRS.

Each procedure consists of a *body* (presented under graphic or text format), which describes the steps of the procedure/plan[5], an *invocation* condition, which specifies the goal the procedure may fulfill (in our example the goal to get the robot to a particular position) or the events to which it reacts, and a *context* describing under which situations the procedure is applicable. Other piece of information are stored in procedure such as facts to conclude or retract upon successful execution (e.g. the `trajectory-history` `EFFECTS` of the procedure Figure 3), documentation or properties which hold user-defined property/value pairs (the `used-resources` and `priority` `PROPERTIES` of the procedure Figure 3).

2.1.1 Goals

In PRS, *goals* are descriptions of a desired state associated to a behavior to reach/test this state.

Achieve: The goal to **achieve** a certain statement `C` is written `(achieve C)` or `(! C)`. So the goal to position the robot is written `(achieve (position-robot 20 10 45))`. Note that this goal is satisfied if the robot is already located at this position (i.e. the database "contains" this information). Note also that the two plans presented above are achieving this particular goal (although in a different context).

Test: The goal to **test** for a statement `c` is written `(test c)` or `(? c)`. So the goal to test if the robot is located in a particular position would be written `(test (position-robot 20 10 45)))`. Similarly, the goal to find out where is currently located the robot is written using variables[6] `(test (position-robot $x $y $theta))`. This implies that either the database contains this information or there is a procedure to find out what is the position[7].

Wait: The goal to **wait** until a statement `c` is true is written `(wait c)` or `(^ c)`. So the goal to wait until the robot is ready to move could be written `(wait (robot-status ready-for-displacement))`. No attempt is made by the plan which posts this goal to make the statement true. Technically, a wait goal never fails (it just sleeps forever). In our example (Figure 2), we wait for a disjunction which insures that the awaited statement will eventually become true.

Preserve: The goal to passively **preserve** `c` is written `(preserve c)` or `(# c)`. Still refering to our example, one could imagine that the system could post a goal such as `(& (achieve (position-robot 20 10 45)) (preserve (~ (robot-status emergency))))` (the ~ is the negation operator). Such goal insures that the condition `(~ (robot-status emergency))` remains always true during the achievement of `(position-robot 20 10 45)`. This behavior is usually refered as a "guarded" action.

preter in C-PRS.

[5] Some procedures, called action procedures, just have an external function call as body.

[6] There are two types of variables in PRS, $ variables are similar to the variables in logical programming language, while @ variables can be rebound in the procedure scope.

[7] Another possibility is to have position declared as an evaluable predicate which would run some internal code to "read" the position from the appropriate place (odometric system, etc).

Maintain: The goal to actively **maintain** `c` is written `(maintain c)` or `(% c)`. So the goal to maintain the robot battery level above 20% while executing a trajectory can be written: `(& (achieve (execute-trajectory @traj))` `(maintain (battery-level 0.200000)))`. Note that if the `maintain` goal fails, the system interrupts the goal to achieve and try to reestablish the maintained goal. In our example, one can imagine that the trajectory execution would be interupted and some attempt would be made to reload the battery.

The `wait (^)`, `preserve (#)` and `maintain (%)` operators can usually be used to implement sophisticated supervision and control operations.

There are two last goals which can be used to explicitely assert or retract information from the database. For a statement `c` they are respectively written `(assert c)` or `(=> c)` and `(retract c)` or `(~> C)`.

2.1.2 Body

The body part of the procedure is built using the different types of goal presented above. As illustrated with our two examples, one can build "text" or graphic procedures. In text procedures, the execution start from the first instruction and proceeds from then, following the standard programming structures such as if-then-else, while, do-while, parallel (//), goto, etc. In graphic procedures, the execution starts from the `start` node and proceeds to the next nodes achieving the goal labelling the edge. In both case, the goal is the "basic" instruction. When a goal appears as the condition of a conditional instruction (such as an if-then-else, a while or before an "if-then-else" node (see node `n4` on Figure 3)) the condition is satisfied if the goal can be achieved, it fails otherwise. On the other hand, when a goal is not in a conditional position, its failure leads to the complete procedure failure.

2.2 Meta-level Procedures

The set of procedures in a PRS application system not only consists of procedural knowledge about a specific domain (Figure 2 and 3), but also includes *meta-level* procedures (Figure 4) – that is, procedures able to manipulate applicable procedures, goals, and tasks of PRS itself. The use of meta-level procedures ranges from methods for choosing among multiple applicable procedures, or to insure mutual exclusion on critical resources. To achieve such objectives, these meta-level procedures make use of information about plans, goals, facts that is contained in the database or in the *properties* slot of the procedure. For example, the meta procedure presented in Figure 4 insures that any fact invoked procedure, is intended. Moreover, it guarantees some kind of priority mechanism if some particular properties hold for the procedure to intend.

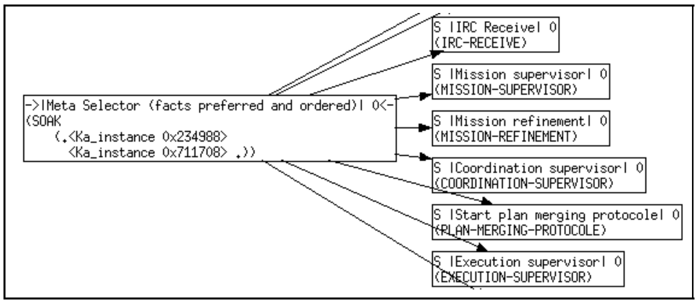

Figure 6: Partial Task Graph Snapshot

2.3 The Interpreter

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

An important part of the main loop is the one which finds applicable procedures and selects those to be intended. 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 procedures, in the light of the concluded facts on the previous set of applicable procedures. This inner loop stops whenever no applicable procedure is found. This means that there exist no more criteria (i.e. applicable meta procedures) to select among the applicable procedures.

The tasks graph holds all the tasks which have been intended. Indeed, when a procedure is applicable, the system (i.e., the main loop or some meta procedure execution) may decide to intend it (i.e., to propose it for execution). If the procedure instance is invoked by an event, it results in a new task, if it was invoked by a goal, it is intended in the task the goal originated. This distinction makes sense as plans pursuing a goal posted by the system would be intended in the same task as the one their goal originated from. On the contrary, procedures applicable because of changes in the world, i.e. events, are not explicitly connected to any previous activity. Figure 6 illustrates a situation where a meta procedure (`Meta Selector...`) is currently executing and adding new tasks corresponding to fact invoked applicables procedure.

3 PRS as a High Level Supervision and Control Language for Mobile Robots

Considering the PRS language and its interpreter, we see that PRS remains a generic tool, which would allow implementation of systems ranging from almost pure rule based system to procedural approach. It provides a general frame for acting on and maintaining a declarative representation of the environment. Nevertheless,

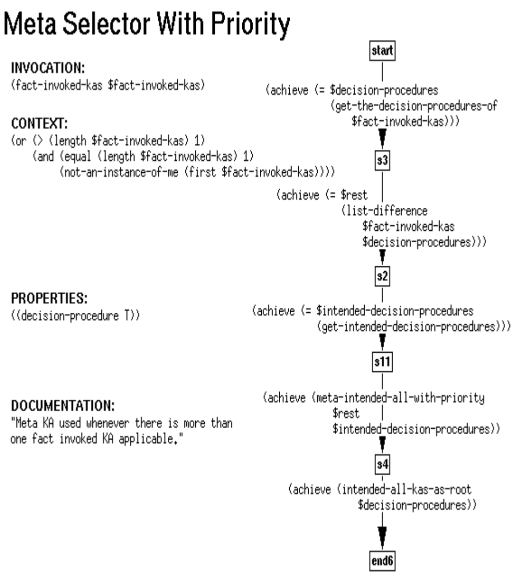

Figure 4: A Simple Meta Level Procedure

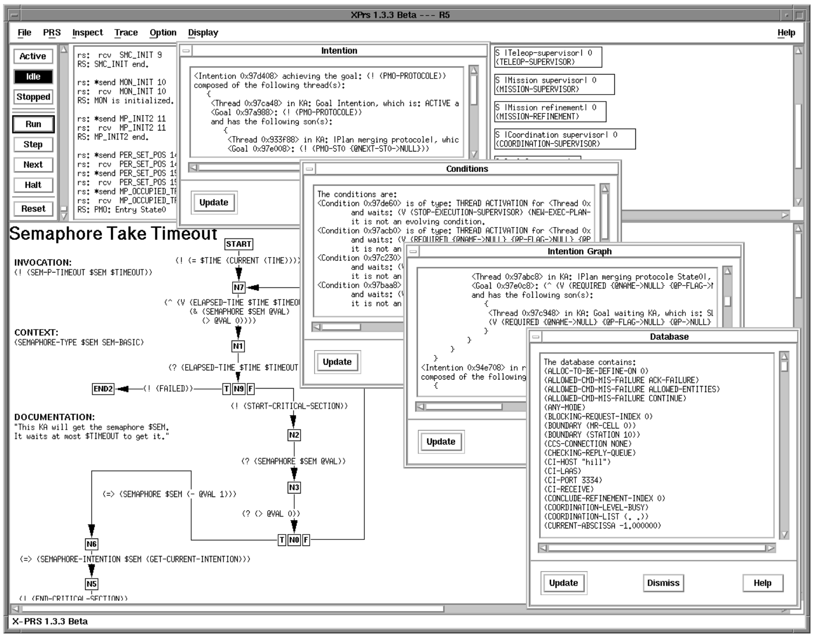

Figure 5: C-PRS Interface (from Martha)

there are a number of reasons why the PRS language appears to be well suited to implement control and supervision components of mobile robots, and we shall now examine them.

3.1 Partial Plan/Script Representation

In PRS, each procedure 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 procedures) the choice of the adequate plan 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 in a partially known and/or dynamic environment.

For example, in the the Eden experiment, 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 (2d or 3d) is left to the interpreter and possibly to meta-level procedures which decides which method is the best in the current situation.

3.2 Event and Goal Driven Behavior

Procedures can be triggered upon occurrence of events or posting of goals. This is a key feature for implementing a periodic monitoring through a set of situation driven procedures while refining and executing a plan as provided by the planner. A convenient way to interface the supervisor and the planner is the PRS data base. This allows for example to express "execution modalities" as facts which will modify the execution of plans, inhibit or awaken others.

The notion of goal in PRS is rather strong as it really represents an objective the interpreter tries 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 unsuccessful attempt). As a consequence, a goal is considered as failed after *all* procedures for each valid unification binding have been tried, and have failed.

The notion of event differs from the notion of goal. An event 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 explicit objective, and the overall success or failure of these event-driven procedures is not analyzed by any means. Moreover, 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.

3.3 Advanced Reasoning

The meta level reasoning available under PRS provides a powerful mechanism to control the PRS main loop. Currently, meta level reasoning is mainly used in the procedure selection part of the PRS main loop[8].

In applications such as the ones developed at LAAS, meta level reasoning has been 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.

In fact, by using meta level procedure, the user gets a hook on the intending process (i.e. which procedures/plans at the end gets a chance to be executed), and therefore gain an important control on the PRS main loop.

3.4 Real-time Aspects

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.

It turns out that if T_{Pars} is the time to parse a procedure invocation, T_{Int} is the time to intend a procedure and T_{Choose} is the time to choose among applicable procedure, the cycle time CT_i depends on the previous cycle time CT_{i-1}, the frequency of event arrival μ_{i-1} during the cycle C_{i-1}, the time taken by the action $T_{Exec}(i)$ (if any) executed during cycle C_i and a constant value $(T_{Int} + T_{Choose})$.

We can show that the cycle time of C-PRS is: $CT_i = (CT_{i-1} \times \mu_{i-1} \times T_{Pars}) + T_{Int} + T_{Choose} + T_{Exec}(i)$
Defining maxima we show that for all i: $CT_i \leq \mu_{Max} \times T_{Pars} \times CT_{i-1} + T_{Exec}^{Max} + T_{Int} + T_{Choose}$
The first observation we can make is that to have a bound on all the CT_i we need: $\mu_{Max} \times T_{Pars} < 1$.
Interpreting this constraint means that if this value $\mu_{Max} \times T_{Pars} \geq 1$ then the value of CT_i diverges and it becomes impossible to guarantee an upper bound on the reaction time. However, if $\mu_{Max} \times T_{Pars} < 1$, then we can show the existence of an upper bound on CT_i which is: $\frac{T_{Exec}^{Max}+T_{Int}+T_{Choose}}{1-\mu_{Max} \times T_{Pars}}$.

See [9] for a more detailed account of these results.

From this bound on reaction time, 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 procedures, it is easy to implement a mechanism which can guarantee that a procedure with a particular property is intended as root of the task graph (therefore executed before any other procedure). The meta level procedure on figure 4 does exactly that for all fact-invoked procedures.

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

3.5 Miscellaneous Features

C-PRS interfaces (see Figure 5) provide mechanisms to the user to examine the various tasks intended, to check/change the content of the database, to follow the execution of particular procedure, and to load or delete procedures on the fly.

C-PRS provides a number of mechanisms to allow interaction with the real world: a communication library over sockets, but also more sophisticated means to provide high level interprocess communications and shared memory services between C-PRS and low level modules [5].

In the Eden experiment, all C-PRS code was still executed on a SparcStation, to which the robot remained connected by an Ethernet cable. In the Martha project, the C-PRS application is small enough (less than 300k) to fit on board the robots on a 68040 boards in a VME rack under VxWorks.

4 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 procedures with a guaranteed bound on their execution time. This could be achieved using compilation techniques similar to the ones used in Kheops [7], 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 plans. Each plan 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 task concept, must be further developed to be easily handled by the user [4]. The activity tree is an important representation level 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.

5 Related Work

The earliest work on using PRS to control mobile robot is a study performed by Georgeff *et al* at SRI and described in [6]. 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 [13], the authors describe an implementation of procedural reasoning (called UM-PRS) to control an outdoor environment vehicle. The paper describes early experiments which although very promising do not reach the level of achieveness obtained with C-PRS in the Eden and Martha experiments.

There are other languages which have been used to program control and supervision system for autonomous mobile robot Esterel[9] [3] and Rex[10]/Gapps [11]. A common point to this two languages is that they used a synchronous approach, which allows the user to prove some temporal properties of the system, as well as make some formal proof of the resulting system. However, we consider these languages as complementary to PRS, which, as we have shown, provide a richer and higher level language, but lacks most of the nice properties which result of the synchronous approach paradigm.

6 Conclusion

This paper presents the use of PRS as a high level language for supervision and control systems for autonomous mobile robots. After a presentation of PRS, we discuss some of the critical features of PRS: its plan (procedure) representation for plan execution and goal refinement; its ability to construct and act upon partial (rather than complete) plans; its ability to pursue goal-directed tasks while being responsive to changing patterns of events in bounded time; its ability to manage multiple tasks in real-time; its default mechanisms to handle real-time demands of its environment; and its meta-level reasoning capabilities have been extensively used in both applications. C-PRS proves to be very flexible and has been used in different implementations of control and supervision systems for mobile robots. Each application is too complex to be presented in this paper but have been described in previous papers: Martha [1] and Eden [12]. In the Eden and in the Martha experiment, the mission interpretation, as well as the control of the functional modules (motion execution, motion planner, perception, etc) are implemented in C-PRS. More specifically, in the Martha project, which involves a fleet of autonomous robots, the plan coordination between robots (Plan Merging Operation [2]) is also implemented in C-PRS. Work on improving some features in C-PRS is on-going to better suit embedded control and supervision of autonomous mobile robots.

[9]Esterel is being used at INRIA.

[10]Rex has been used on SRI's Flakey robot.

References

[1] L. Aguilar, R. Alami, S. Fleury, M. Herbb, F. F. Ingrand, and F. Robert. Ten autonomous mobile robots (and even more) in a route network like environment. In *Proceedings of IROS 95*, Pittsburg (USA), 1995.

[2] R. Alami, F. Robert, F. F. Ingrand, and S. Suzuki. Multi-robot cooperation through incremental plan-merging. In *IEEE ICRA, Nagoaya, (Japan)*, 1995.

[3] G. Berry and G. Gonthier. The Synchronous Programming Language Esterel: Design, Semantics, Implementation. *Science of Computer Programming*, 19(2), 1992.

[4] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. Integrated planning and execution control of autonomous robot actions. In *IEEE ICRA, Nice, (France)*, 1992.

[5] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE ICRA, San Diego California, (USA)*, 1994.

[6] M. P. Georgeff, A. L. Lansky, and M. Schoppers. Reasoning and planning in dynamic domains: an experiment with a mobile robot. TN380, AI Center, SRI International, Menlo Park, California (USA), 1987.

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

[8] F. F. Ingrand, R. Chatila, and R. Alami F. Robert. Embedded control of autonomous robots using procedural reasoning. In *Proceedings of ICAR 95*, San Feliu de Guixols, Spain, 1995.

[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 IJCAI*, Saint Paul, Minnesota (USA), 1988.

[12] S. Lacroix, R. Chatila, S. Fleury, M. Herrb, and T. Simeon. Autonomous navigation in outdoor environment : Adaptative approach and experiment. In *IEEE ICRA*, San Diego, California, May 1994.

[13] 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 IRFFSS*, Houston, Texas (USA), March 1994.