Dependability Issues in a Robot Control Architecture

Rachid Alami, Raja Chatila , Félix Ingrand, Frédéric Py[*]
LAAS/CNRS,
7 Avenue du Colonel Roche, F-31077 Toulouse Cedex 04, France
{*rachid,raja,felix,fpy*}*@laas.fr*

Abstract

This paper presents the LAAS architecture for autonomous mobile robots and some recent developments to improve the dependability of the system. In particular, this paper focuses on the role of the Execution Control level of this architecture. This level has a fault protection (safety bag) role with respect to the commands issued to the functional level which is connected to the physical devices. These commands come either from the decisional level, or from the functional level itself. We introduce a new approach and a new tool inspired from the model checking domain. We present a new language to specify the model of acceptable and required states of the system (valid contexts for requests to functional module and resources usage). This language is compiled in an OBDD (Ordered Binary Decision Diagram) like structure which is then used online to check the specified constraints in real-time. Such a model checking approach could be extended to check off line more complex temporal properties of the system.

1 Introduction

There is an increasing need for advanced autonomy in complex embedded real-time systems such as robots, satellites [Powell and Thévenod-Fosse, 2002], or UAVs. The growing complexity of the decision capabilities of these systems raises a major problem: how to prove that the system is not going to engage in dangerous states? How to guarantee that the robot will not grab an object with its arm, while moving in a crowded environment (which could supposedly arm somebody)? How to make sure that a cleaning robot does not vacuum the cat? Or that a robot walking assistant helping elderly people does not speed up when a person is walking leaning on it?

There are mechanical designs and physical securities to prevent some of theses to happen but still, in advanced autonomous systems, most of these actions result from explicit decisions from the decisional level. So a partial response to this problem is to use a planner which only synthesizes valid and safe plans. Yet, high level planners do not (cannot) have a complete model representing the full extend of their actions. Some of these actions are refined by the supervisor/executive, therefore the particular sequence of low level commands sent to the physical system is not completely under the control of the planner. Moreover, other plans, given as scripts or procedures, are often used to perform some task refinement. These scripts and procedures are usually written in high level language providing concurrent actions executions. In most cases, this concurrency results in unforeseen and unplanned interactions.

A solution to guarantee the safety properties is to integrate in the architecture a system that formally controls the validity of the "low level" commands sent to the physical system and prevents it from entering an unsafe state. This controller must check the system consistency during execution without affecting the system basic functionalities, such as reaction time.

The LAAS[1] architecture, presented in section 2, foresaw such a mechanism in its Execution Control Level. Section 3 presents the Execution Control Level roles and requirements, with a state of the art and related works. Section 3.2 gives an informal description of the pro-

[*]List of authors in alphabetical order.

[1]LAAS Architecture for Autonomous System.

posed approach, the tool and the language we use for the Execution Control Level. In section 3.4, we present some experimental results of the execution control system. Section 4 mentions other aspects of the LAAS architecture and tools which participate to the overall dependability of the system.

2 The LAAS Architecture

The LAAS architecture [Alami *et al.*, 1998] was originally designed for autonomous mobile robots. This architecture remains fairly general and is supported by a consistently integrated set of tools and methodology, in order to properly design, easily integrate, test and validate a complex autonomous system. As shown on Figure 1, it has a number of levels, with different temporal constraints and uses different data representations. These levels are:

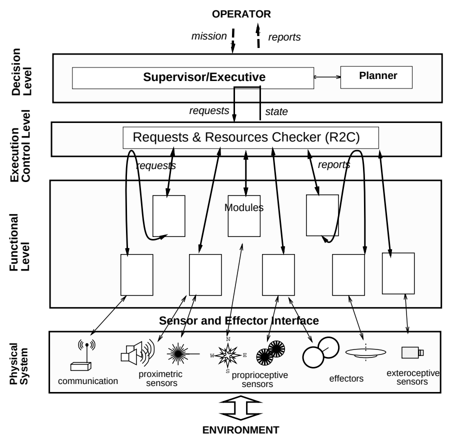

Figure 1: The LAAS Architecture.

- *The decision level:* This higher level includes the deliberative capabilities such as, but not limited to: producing task plans, recognizing situations, faults detections, etc. It embeds at least a supervisor/executive, which is connected to the underlying level, to which it sends requests that will ultimately initiate actions and start treatments. It is responsible for supervising plans or procedures execution while being at the same time reactive to events from the underlying level and commands from the operator. Then according to particular applications it may integrate other more complex deliberation capabilities, which are called by the supervisor/executive when necessary. The temporal properties of the supervisor are such that one guarantees the reaction time of the supervisor (i.e. the time elapsed before it sees an event), but not much can be said for other decisional components.
- *The functional level:* Located at the lowest level, it includes all the basic built-in robot action and perception capabilities. These processing functions and control loops (image processing, motion control, ...) are encapsulated into controllable communicating modules. Each module provides a number of services and treatments available through requests sent to it. Upon completion or abnormal termination, reports (with status) are sent back to the requester. Note that modules are fully controlled from the decisional level. Modules also maintain so called "posters"; data produced by the modules, such as the current position and speed (from the locomotion module) or current trajectory (from the motion planning module) which can be seen by other modules and the levels above. The temporal requirements of the modules depend on the type of treatments they perform. Modules running servo loop (which have to be ran at precise rates and intervals without any lag) will have a higher temporal requirement than a motion planner, or a localization algorithm.
- *The execution control level:* Located in between the two previously presented levels, the Requests and Resources Checker (R2C) checks the requests sent to the functional modules (either internally or from the higher level), as well as the resources usage. It is synchronous with the underlying functional modules, in the sense that it sees all the requests sent to them, and all the reports coming back from them. It acts as a filter which allows or disallows requests to pass, according to the current state of the system (which is built online from the past requests and past reports) and according to a formal model (given by the user) of allowed and forbidden states of the functional system. The temporal requirements of this level are hard real-time. This is the level on which this paper focuses.

This architecture naturally relies on several representations, programming paradigms and processing approaches meeting the precise requirements specified for each level. We developed proper tools to meet these specifications and to implement each level of the architecture: IxTeT a temporal planner [Ghallab and Laruelle, 1994], Propice a procedural system for tasks refinement and supervision/executive [Ingrand *et al.*, 1996], and G^{en}$_{o}$M for the specification and integration of modules at that level [Fleury *et al.*, 1994]. These various tools share the same namespace (i.e. the name of the modules, requests, arguments and posters).

This paper focuses on the Execution Control Level and a newer approach (the Requests and Resources Checker (R2C) and tool (ExᴼGen) used to implement it.

3 Execution Control Level for a Dependable Autonomy

The main role of the Execution Control Level and its main component the R2C is a fault protection role. It is the "safety bag" of the LAAS architecture. Faults are inevitable, even more with complex decisional system partially based on informal methods and tools. Yet to be able to use such advanced decisional tools and take advantage of the high level of autonomy they provide, one needs to design systems which prevent the system from engaging into disastrous situations. Thus the execution control level has a "simple", yet critical role in the architecture: to check the system consistency and reject or abort requests threatening the system safety.

- As an interface with all the modules of the functional level, it ensures that all the requests passed to the functional module remain consistent with respect to a model of desirable or undesirable states of the system. For example, it would be the R2C role to make sure that a request to move the robot is not issued while the arm is moving.
- It manages the resources of the system and guarantees that any request leading to an overconsumption or inconsistent use of resources is properly handled.
- It acts synchronously with the functional level to ensure a consistent view of the state of functional modules.
- It acts in guaranteed real-time. No request to the functional level should be delayed more than one R2C cycle before being processed.

This critical role requires the use of formal tools to validate it.

3.1 Brief State of the Art in Execution Control

Many of the concerns raised in the previous section are not new, and some robotics architectures address them in one way or another.

Indeed, some of the requirements presented above were clearly fulfilled by a previous version of the LAAS execution control layer based on KHEOPS [de Medeiros *et al.*, 1996]. KHEOPS is a tool for checking a set of propositional rules in real-time. A KHEOPS program is thus a set of production rules ($condition(s) \rightarrow action(s)$), from which a decision tree is built. The main advantage of such a representation is the guaranty of a maximum evaluation time (corresponding to the decision DAG depth). However, the KHEOPS language is not adapted to resources checking and appears to be quite cumbersome to use.

Another interesting approach to prove various formal properties of robotics system is the ORCCAD system [Espiau *et al.*, 1995]. This development environment, based on the Esterel [Boussinot and de Simone, 1991] language provides some extensions to specify robots "tasks" and "procedures". However, this approach does not address architecture with advanced decisional level such as planners.

In [Rutten, 2001], the author presents another work related to synchronous language which has some similarities with the work presented here. The objective is also to develop an execution control system with formal checking tools and a user-friendly language. This system represents requests at some abstraction levels (no direct representation of arguments nor returned values). This development environment gives the possibility to validate the resulting automata via model-checking techniques (with Sigali, a Signal extension).

In [Goldman and Musliner, 2000], the authors present the CIRCA SSP planner for hard real-time controllers. This planner synthesizes off-line controllers from a domain description (preconditions, postconditions and deadlines of tasks). It can then deduce the corresponding timed automaton to control the system on-line, with respect to these constraints. This automaton can be formally validated with model checking techniques.

In [Simmons *et al.*, 2000] the authors present a system which allows the translation from MPL (Model-based Processing Language) and TDL (Task Description Language) to SMV a symbolic model checker language. Compared to our approach, this system seems to be more designed for the high level specification of the decisional level, while our approach focuses on the online checking of the outcomes of the decisional level.

The new IDEA system [Muscettola *et al.*, 2002] developed at NASA Ames, following the RAX-PS DS1 experiment, is also addressing robust execution control. But it is taking a strong temporal constraints approach to check online and in real-time that the actions planned are consistent with its temporal constraints network.

3.2 The Request and Resource Checker.

The R2C is designed to support safe execution control of the system. It has been designed to address the requirements introduced in section 3. In figure 2, we present its general organization.

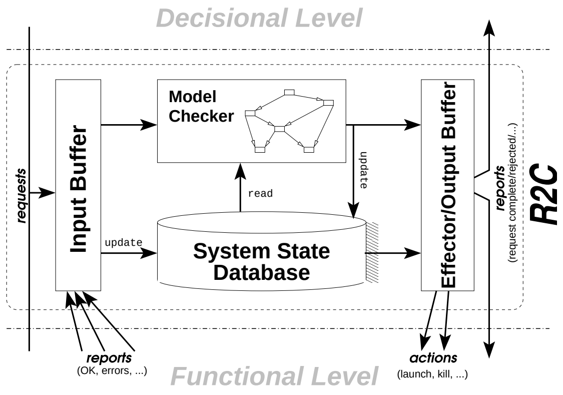

Figure 2: R2C general view.

The R2C includes :

An Input buffer : this buffer will capture all the events coming from the system to treat them during next R2C cycle. The events may come from :

- The decisional level. These events are generally requests of services for the functional level.
- The functional level. These events are generally reports of ending services and resources modifications. But we can also have requests of services made from a module to another module. Indeed the R2C has to be omniscient, it has to know precisely the system state (i.e. all active and past requests/reports).

A System State Database : this component maintains the system state view according to the events. This view describes:

- The active requests (with their arguments).
- The past requests (with the arguments and returned values of the last correctly executed instance of a request).
- The resource levels.

A Model Checker This component is the reasoning part of R2C. This is where the R2C will decide which actions to do according to the newly captured requests/reports and the current system state. It is based on model checking techniques and have real-time guarantees.

An Effector This component will execute deduced actions in the system. These can be sent to :

- The *decisional level.* The events sent to this level are generally report messages (the result of an execution or the reason of the rejection of a particular service).
- The *functional Level.* These events may be new demands of services or requests to abort a particular service. Of course we can also find here reports to a requester from the *functional level.*

For each execution cycle, the *input buffer* captures events from the decisional or the functional level, it updates the *system state database* then, according to these new events and the new state of the system, the *model checker* decides the actions to perform to keep the system consistent with its model. Its actions are sent to the *effector* and to the *system state database* to maintain a correct system view. The effector then sends the appropriate events to the decisional and functional levels.

3.3 Deducing Safe Actions

The most critical part of R2C is the *model checker*. This component deduces in real-time the safe actions to do according to incoming events and to the current state. To implement the *model checker* we have defined the ExᴼGen language [Ingrand and Py, 2002] to describe the inconsistent states the user wants the system to avoid.

The *model checker* is based on model-checking techniques. It represents its deducing part via an OBDD[2] equivalent data structure named Ordered Constrained Rule Diagram (OCRD [Ingrand and Py, 2002]). Such a kind of data structure has multiple advantages :

- An OBDD is a Directed Acyclic Graph (DAG) representing a logical formula. So its traversal has a maximum execution time corresponding to its depth.
- This data structure represents formula in a canonical form so it is a compact form avoiding redundancies.
- Another strong point for OBDDs is the fact that this data structure is used for formal model validation via model checking. So we can check the validity of the generated automata according to safety requirements of the system.

3.4 Experimental Results

We implemented an R2C on our XR4000 Nomadics: Diligent (see Figure 3). In its current configuration, Diligent has the architecture presented on Figure 4. There are currently only few rules but the results are quite encouraging.

With an ExᴼGen declaration containing 6 binary `mutexes` between requests, one ternary `mutex` and one `fail` context for a particular request, we build the resulting OCRD in 67 seconds[3] including 40 seconds of diagram size optimization. Indeed when the OCRD – like the OBDD – size is highly dependent on the predicates order, we have included an optimization of the resulting diagram based on the sifting algorithm (see [Rudell, 1993]) adapted to the OCRD data structure.

The resulting OCRD has a maximum depth of 17 and the maximum node traversal number before a controllable one is 13. Therefore, at best, the resulting R2C will make 13 tests before doing any action and the maximum delay of execution of the R2C corresponds to the traversal of 17 nodes (i.e. 17 tests/actions). After simplification (removing of the non reachable branches and choice of one execution branch when two or more are possibles to make it determinist) the resulting OCRD contains 552 nodes.

On our Nomadics XR4000, the average traversal time of the decision diagram in the R2C is around 50 microseconds.

3.5 Toward a safe execution control development tool

The main objective of this work is to provide a development tool for safe execution control. So far, we have developed :

- A user-friendly language (ExᴼGen) for developers to specify the constraints of the system and the ways to avoid inconsistency,
- A compiler generating the deducing part of the R2C according to the specification written in ExᴼGen.
- Other non-application specific components of the R2C.

This is already sufficient to guarantee some safety constraints on the system. Indeed the resulting OCRD is logically equivalent to the constraints expressed by the user using ExᴼGen.

Nevertheless, model checking is a promising choice for further formal automatic verifications of the models. Therefore, future works on the *Execution Control Level* will be aimed at extending the model given by the user and the property one can check (on line or off line) such as temporal logic properties (reachability of a state from a particular state, check for possibly deadlock states. . .).

[2]OBDD : Ordered Binary Decision Diagram.

[3]The machine used for compilation is a Sunblade 100 with 512 Mb of memory.

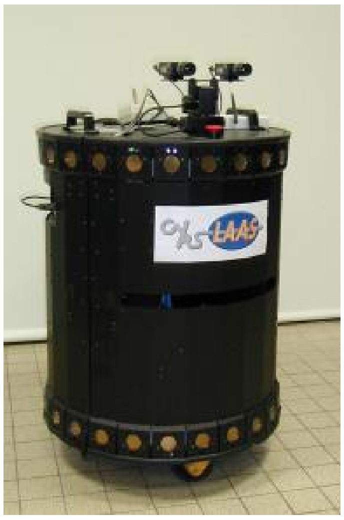
Figure 3: Diligent

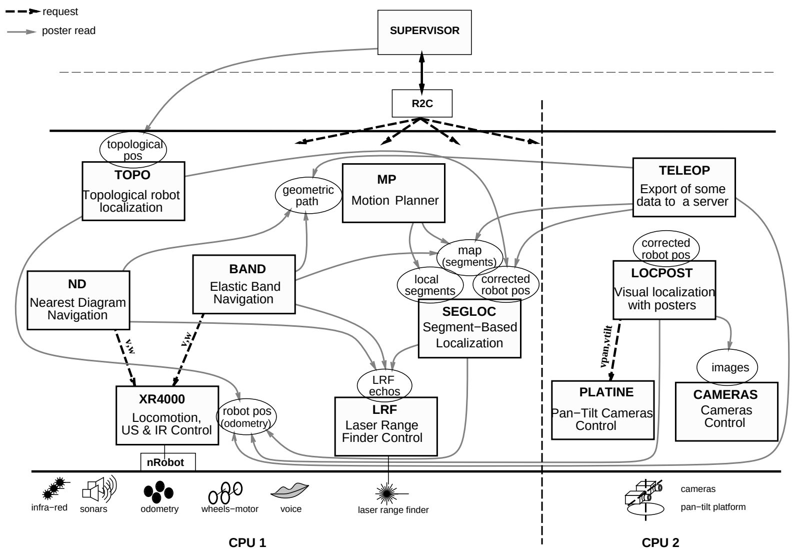

Figure 4: Diligent Architecture

4 Other Dependability Aspects of the LAAS Architecture

If one considers the two other levels of the LAAS architecture, one can identify a number of features and properties which also provide an improved dependability of the overall system.

4.1 The Functional Level

The dependability aspect of the Functional Level mainly relies on the use of a software development environment which enforces a unified development methodology and interface: The Generator of Modules GenoM [Fleury *et al.*, 1994].

GenoM is a tool to design and build real-time distributed software systems. It allows the developer to easily encapsulate operational functions (algorithms broken down into elementary code chunks called "codels") on independent communicating modules. The functions can be dynamically started, interrupted or (re)parameterized upon asynchronous requests sent to the modules by any other element in the system, including other modules, the execution controller or the supervisor. A final reply that qualifies how the request has been executed by the module is sent back. A module operation can be modeled by a finite state automaton.

Thus the modules generated using GenoM are "standardized" servers. Programming a module does not require knowing about the on-board operating system, nor communication or other real-time procedures (synchronization...). Modules are automatically produced by GenoM using a template. Such a systematic specification guarantees coherence in the module design (different modules are usually programmed by different persons) and allows automating their integration. Moreover, all the modules being based on a template, only the codels and the particular activity state automata need to be specified and eventually proven to improve the robustness and dependability, while the largest part of the code is generic and only needs to be tested and qualified "once for all".

4.2 The Decisional Level

This subject of "decisional autonomy bringing some improved dependability" in the system is discussed at length in a companion paper [Alami, 2002], nevertheless, we can sketch here the main ideas it discusses.

The LAAS control architecture is plan-based: this means that the planner/supervisor pair is built on an explicit reasoning on the tasks and on the robot capacities to achieve them in a given context, and even in a given time.

The ability to plan and then to use various ways to perform a task through context dependent refinement, enhances the robot dependability because it enlarges the spectrum of contexts when and where the robot is able to perform its task and/or to detect autonomously its incapacity to perform it.

When the supervisor fails to recover from a problematic situation, it can call the planner with an updated description of the world and it can request it to produce a new plan taking into account the previous failure and the new situation.

The robot supervisor is programmed in Propice (formerly PRS) [Ingrand *et al.*, 1996]. It provides an interface with all the robot modules thus allowing to program robot control. All supervisor activities (task refinement, control, display, dialog. . .) are programmed using goal directed and/or situation driven procedures.

The planner is programmed in IxTeT [Ghallab and Laruelle, 1994] which is a partial order planner using a temporal logic.

5 Conclusion

We have briefly presented the LAAS architecture, its components and its integrated tools. Then the presentation focuses on the Execution Control Level of this architecture and its main component, the Requests and Resources Checker (R2C). This layer of the LAAS architecture has a critical role with respect to safety and faults protection. It must guarantee that the functional modules which "act" on the real physical system are properly controlled and do not engage in "dangerous" situations with respect to the "commands" they are executing. As a result, this component of the LAAS architecture has a critical role, with respect to dependability.

The R2C and its associated tool ExoGen propose a language in which the user can specify, in the GenoM namespace, the contexts in which a particular request (with constraints on its arguments) can be executed.

Our approach uses OCRD, a structure similar to OBDD. Moreover, it offers some real-time properties such as the guaranty of the maximum time taken to check new requests. Another interesting property inherited from OBDD is the reduction of input formula to a canonical form. The generated DAG is relatively compact (depending on predicates ordering method) and is unique. The counterpart of this reduction property is that the completeness checking of a declaration (that all states are specified) is harder than with a complete representation.

Other dependability aspects of the LAAS architecture have also been discussed:

- The decisional aspects which provide a new "framework" to program dependable and robust plans [Alami, 2002].
- The use of software development tools such as GenoM which provide a generic and high level environment to write functional modules [Fleury *et al.*, 1994].

Although the LAAS architecture already provides some interesting features with respect to dependability, one of our goals is to further improve it in this domain. These extensions may rely on more formal temporal models of the functional modules (by extending the finite automaton of the module activities), as well as a tighter interaction between the temporal planner and the executive/supervisor.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming*, 17(4):315–337, April 1998.

[Alami, 2002] R. Alami. Robot decisional aspects for improved dependability. In *Proceedings of the 2nd IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*, October 2002.

[Boussinot and de Simone, 1991] F. Boussinot and R. de Simone. The ESTEREL Language. *Proceeding of the IEEE*, pages 1293–1304, September 1991.

[de Medeiros *et al.*, 1996] A. D. de Medeiros, R. Chatilla, and S. Fleury. Specification and Validation of a Control Architecture for Autonomous Mobile Robots. In *IROS*, pages 162–169. IEEE, 1996.

[Espiau *et al.*, 1995] B. Espiau, K. Kapellos, and M. Jourdan. Formal verification in robotics: Why and how. In *The International Foundation for Robotics Research, editor, The Seventh International Symposium of Robotics Research*, pages 201 – 213, Munich, Germany, October 1995. Cambridge Press.

[Fleury *et al.*, 1994] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation*, Atlanta, USA, 1994.

[Ghallab and Laruelle, 1994] M. Ghallab and H. Laruelle. Representation and Control in Ixtet, a Temporal Planner. In *Proceedings of AIPS-94*, pages 61–67, 1994.

[Goldman and Musliner, 2000] R. P. Goldman and D. J. Musliner. Using Model Checking to Plan Hard Real-Time Controllers. In *Proc. AIPS Workshop on Model-Theoretic Approaches to Planning*, April 2000.

[Ingrand and Py, 2002] F. Ingrand and F. Py. An Execution Control System for Autonmous Robots. In *IEEE International Conference on Robotics and Automation*, Washington DC USA, May 2002.

[Ingrand *et al.*, 1996] F.F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation*, Mineapolis, USA, 1996.

[Muscettola *et al.*, 2002] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt. Idea: Planning at the core of autonomous reactive agents. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, October 2002.

[Powell and Thévenod-Fosse, 2002] D. Powell and P. Thévenod-Fosse. Dependability issues in ai-based autonomous systems for space applications. In *Proceedings of the 2nd IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments*, October 2002.

[Rudell, 1993] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *IEEE/ACM ICCAD'93*, pages 42–47, 1993.

[Rutten, 2001] E. Rutten. A framework for using discrete control synthesis in safe robotic programming and teleoperation. *IEEE International Conference Robotics & Automation*, pages 4104–4109, May 2001.

[Simmons *et al.*, 2000] R. Simmons, C. Pecheur, and G. Srinivasan. Towards automatic verification of autonomous systems. In *IEEE/RSJ International conference on Intelligent Robots & Systems*, 2000.