Investigating software dependability mechanisms for robotic applications

Laurent, T., Nana

Abstract*—this paper relates our case study of the application of software dependability to the design and implementation of robotics languages and architectures. The failure of a robot or that of its control system may lead to important damage to the environment in which it evolves or on human life. On the other hand, the amount of software used in robotics systems is increasing radically. For these reasons, solutions for software safety are needed in order to meet the dependability requirement inherent to such systems. In this paper, we show that in robotic systems, formal specification and verification methods are only used in a few cases and fault tolerance mechanisms as well as software testing techniques are exploited little. We have experienced static and dynamic testing as well as specification and verification with Petri nets in the context of the robotic language PILOT (Programming and Interpreted Language Of actions for Telerobotics) and its control system software. Among the benefits of the application of those approaches to the control system software, we can quote: detection and elimination of faults, fault tolerance capabilities, algorithms simplification, operator assistance in the design of plans, security of modification in the course of mission execution.

Index Terms—*Robotics languages, robotics architectures*, *software dependability*

1. Introduction

NOWADAYS, robots, like the tasks they are assigned, are more and more complex. On the other hand, important robotics projects may involve hundreds of engineers and/or researchers and several institutions. In such a context, the availability of proper architecture is necessary in order to structure the different levels of abstraction and to improve reusability and modularity of the hardware and software components of robotics systems.

Most robots are also critical systems. For example, the failure of a mobile robot or that of its control system, may lead to important damage to the environment in which it evolves or on human life. In the same way, in specific applications such as the nuclear field, remote surgery, or search and rescue missions, the failure of the robot may have disastrous consequences on human life.

The amount of software used in robotics systems is increasing rapidly. As a consequence, the use of software engineering techniques becomes necessary and even mandatory from the first steps of the realization of robotic projects in order to reduce re-designing costs and to facilitate software evolution and maintenance. These techniques include dependability means which are of particular interest for critical systems.

In this paper, we focus on software dependability in robotic architecture. Dependability is a justified guarantee that the system will appropriately perform its mission and not cause a catastrophe. Two means of dependability are often used: *fault avoidance* and *fault tolerance* [1]. Fault avoidance aims at producing a system containing as few faults as possible. To achieve this goal, the mechanisms used are formal methods and languages of specification and testing. As for fault tolerance, it aims at making it possible to the system to behave in a satisfactory way even in the presence of faults. *Fault forecasting* is also a common approach used for dependability. It aims at estimating the present number, the future incidence and the likely consequences of faults.

The second section of this paper is dedicated to the problem statement. The work related to software dependability in robotic architecture is presented in the third section. The language PILOT and its software architecture, both proposed by the LISyC Laboratory of Brest in France for safe robotic applications, are presented in the third section. The fourth section addresses the solutions proposed for the dependability of robotics applications in the context of PILOT and its software architecture. The results of the application of these solutions are presented in the fifth section. The seventh section is dedicated to the discussion about the solutions proposed or experienced within the framework of PILOT and those provided by other robotic architectures. This paper ends by conclusions and perspectives in the eighth section.

2. Problem statement

Although various solutions have been proposed for general purpose software dependability, most of them are currently unapplied or applied in very few cases in the field of robotics. This is probably due to the need of adaptation for a proper use in robotics. Another reason is the low implication of computer science specialists in the early steps of the design of robotics systems. Finally, some aspects of the software engineering process such as testing are not considered as important as they should be.

The following questions are therefore of particular importance. Are the existing software dependability solutions adapted to the field of robotics? How to adapt existing software dependability solutions to the potential designers/developers of robotic systems?

In order to find an answer to the above questions, we have investigated the application of software

Manuscript submitted September 10, 2006.

L. Nana is with the Computer Science Department, University of Brest, CS 93837 – 29238, Brest Cedex 3, France (e-mail: nana@univ-brest.fr).

dependability to the design and implementation of robotic control architecture.

3. RELATED WORK

The modularity inherent to robotic architectures prevents faults by simplifying the development of their components and their testing. It also allows a better error confinement and as a consequence, better fault tolerance capabilities. Apart from this intrinsic feature, the taking into account of software dependability in the design and the implementation of robotic architecture still remains limited nowadays. A certain number of studies however were carried out in this field for fault avoidance and fault tolerance. As far as fault forecasting is concerned, few studies have been done. A short overview of robotics architectures will be given first, and then the state of the art of software dependability in those architectures will be presented.

3.1 Brief overview of robotic architecture

Robotics architectures can be classified in four main categories: *centralized architectures*, *hierarchical architectures*, *behavioral architectures* and *hybrid architectures*.

Centralized architectures were inspired by artificial intelligence [21]. They place planning in the centre of the system and share the axiom whereby the central problem in robotics is cognition, i.e., the handling of symbols to maintain and act on a model of the world, the world being the environment with which the robot interacts. Among centralized architectures, we can mention: the planning system STRIPS [22] and Blackboard architecture [9]. Centralized architectures are well adapted for tasks where reactivity and reflex are not essential criteria.

Hierarchical architectures break up the programming of the applications into increasingly abstract levels. The role of each level is to break up a task recommended to it by the higher level, into simpler tasks which will be ordered at the lower level. The highest level manages the global objectives of the application, whereas the lowest level orders the actuators of the robot. The best known example of this type of architecture is NASREM [15]. In the same family, we can quote the architecture of the LIFIA [8] and architecture SMACH [31]. Hierarchical architectures are adapted for tasks which are carried out in a foreseeable environment and which require a high precision. Hierarchical architectures generally have a rather low reactivity.

Behavioural architectures were born with Brooks's subsumption architecture [5]. They are based on the idea that a complex and evolved behaviour of a robot can emerge from the simultaneous composition of several simple behaviours. The architecture DAMN suggested by Rosenblat [26] is a variation on the work of Brooks. The main asset of behavioural architectures is their speed of reaction vis-à-vis the external events or to specific situations. However, they are not adapted to applications which involve the competition of several behaviours (difficulty to ensure the stability of execution of the law of complex orders) or impromptu changes of the mission (the behaviours are pre-established).

Hybrid architectures combine the reactive capacities of behavioural architectures and the capacities of reasoning specific to hierarchical architectures. Among them, we can cite: the CONTROLSHELL [28], the architecture of the LAAS [24] or architecture ORCCAD [11]. After this overview of robotic software architecture, the next subsection presents the state of the art of dependability in robotic architectures.

3.2 State of the art of dependability in robotics architectures

3.2.1. Fault avoidance

Formal verification

In the subsumption architecture, behaviours are modelled by augmented finite state machines. This modelling allows the application of formal verification methods. The architecture of the LAAS integrates an execution control component which has been synthesized from a model of acceptable and dangerous states using model-checking techniques. Its aim is to ensure that the system will never reach an inconsistent state [24]. In the architecture ORCCAD, the synchronous language ESTEREL and formal verification tools are used for the specification and verification of the control part [33]. In ProCoSA, an environment for implementing advanced mission management in autonomous vehicles, behaviours are formally described with Petri nets [2][3]. Running the system is achieved by an automaton which performs requests to functionality servers in accordance with the specification encoded in the Petri nets. Rutten proposed a toolkit for the safe programming of robotics applications [27]. The latter is based on the synthesis of controllers [25].

Testing

Very few reports exist on the application of testing to robotics software. However, an intensive testing was carried out in the context of the RAX architecture [4][13]. The authors underline the relevance of intensive testing, notably the problem of defining suitable test oracles.

3.2.2. Fault tolerance

Fault tolerance usually involves 4 main steps: *error detection*, *damage evaluation*, *error recovery* and *system repair*. We will only consider error detection and error recovery which are the main steps currently addressed in the context of robotics architectures.

Error detection

In RoboX9 [32], error detection is implemented with *timing checks* by using watchdogs. These checks are used for speed monitoring, obstacle avoidance, bumpers and laser sensors. RoboX9 also implements *reasonableness checks* to monitor robot speed, in order to ensure that this remains within a specified interval. In the R2C component of the LAAS architecture [24], error detection is implemented using *safety-bag-checks* which consist in intercepting and blocking the system commands if they do not respect a set of safety properties specified during development. Error detection is also performed with *Monitoring diagnosis* which consists in checking system behaviour against a mathematical model. It is mainly used to detect hardware errors. In the European project ADVOCATE II, an intelligent diagnosis system has been proposed for a deepened diagnosis of AUV (Autonomous

Underwater Vehicles) in case of failure [23]. This system is based on the following artificial intelligent techniques: Bayesian belief networks, fuzzy logic and neuro-symbolic systems. The architecture proposed in [29] by Seabra Lopes *et al.* for the assembly of tasks also provides at various levels of abstraction, functions for the diagnosis of error. The modelling of the failures of execution made through taxonomies and causality networks plays a central part in the diagnosis.

Error recovery

The main solutions used to implement error recovery in robotics systems are *error confinement*, *positioning in a safe state*, and *reconfiguration*.

In RoboX9, *error confinement* is implemented by using dedicated processors. Critical tasks such as localization and navigation tasks are executed on one processor and the other tasks are launched on another processor.

Positioning in a safe state may be executed in case of critical component failure or while executing a time-consuming recovery action. Such mechanisms are provided for example on the ASTER'x AUV of the IFREMER [16] (parking at the bottom of the sea, procedure of safe failure to release the robot from the ballast, etc.).

Software reconfiguration is not directly addressed in most robotics architectures. Nevertheless, one can quote corrective maintenance through the use of patches, which is a particular form of reconfiguration. It has been applied to recover the Martian rovers Pathfinder and Sojourner which failed respectively because of priority and RAM management failures.

Another implementation of fault tolerance found in robotic architecture is the *fault masking technique* [1]. It consists in taking advantage of the redundancy resulting from the combination and permutations of possible actions to replace a combination of actions which have become unusable, due for example to a failure, by an alternate combination of actions. Such technique is used in architectures such as that of the LAAS, ORCCAD, and CIRCA [7].

Security aspect in the realization of robotics missions was one of the main motivations of the project "Software Architectures for Mobile and teleoperated robotics" initiated by the LISyC Laboratory in the 90's, which led to the creation of PILOT and to its control system. PILOT system has been built as a hierarchical architecture with two main parts: a high level part located on the remote control computer and a low level part embedded on the teleoperated robot. In the two following sections, we present the work completed in this context for the reliability of robotics applications. A short description of the language PILOT and of its control system software is first carried out. The solutions for safety implemented within the framework of PILOT system are then approached.

4. SHORT OVERVIEW OF THE LANGUAGE PILOT AND ITS CONTROL SYSTEM SOFTWARE

4.1. The language PILOT

PILOT is a graphical and interpreted language mainly dedicated to the remote control of robots. It is based on the notion of action. An action encapsulates an order executable by the robot, a precondition and one or more supervision rules to which processing are associated. Two kinds of actions are discriminated: elementary and continuous actions. Unlike a continuous action whose end is triggered by an enclosing primitive of the language, an elementary action generally ends when its predefined goal is reached. Whatever kind it may be, an action is only executed when its precondition rule is true (unless the operator decides to force the execution). In the same way, if during the execution of an action, one of its supervision rules becomes true, then the corresponding processing is performed.

The language PILOT provides control structures for plan building: sequence, conditional, iteration, parallelism and pre-emption. Detailed information on the language PILOT can be found in [6, 14].

4.2. The control system software of PILOT

The control system software of PILOT (FIG. 1) is the interface between the user and the controlled machine (target robot). It comprises six modules: a *Man Machine Interface* (MMI), a *Communications server*, a *Rules Generator*, an *Evaluator*, an *Execution Module* or *Driver* and an *Interpreter*. These modules run in parallel and communicate through sockets and shared memory. The control system can work either in centralized mode, or in distributed mode. The choice of the working mode is done in a static way (before compilation).

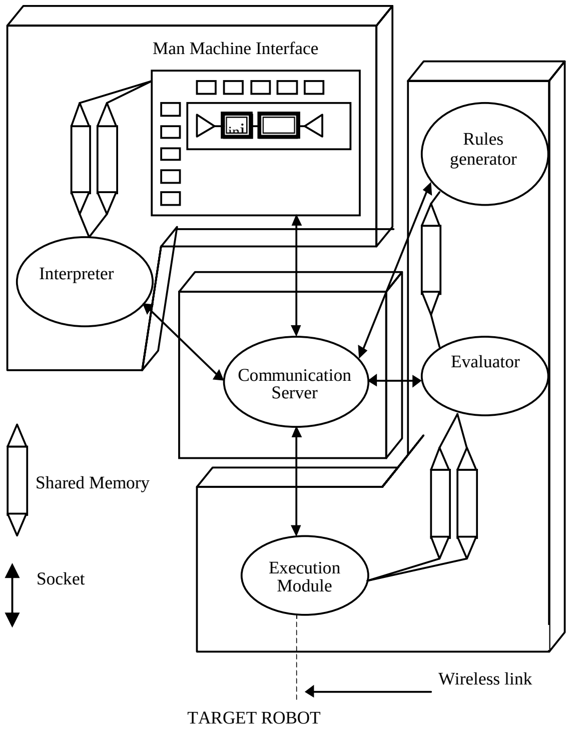

FIG. 1 The control system of PILOT

The MMI provides means for the construction of plans, the dynamic creation of actions (without recompiling of the code), and the modification of the plan before and during the execution of this last. It also integrates means for the supervision of the execution of the plan. The MMI stores the plan in a storage area shared with the interpreter. This reads the plan in shared memory and sends orders (request for evaluation of precondition of an action, order of starting of the action, etc) to the other modules in order to carry out the execution of the plan. The communications server manages the communications between the modules of the control system. The role of the rules generator is to transform the character strings of preconditions and supervision rules into binary trees. It stores the result in a storage area shared with the evaluator. This evaluates the precondition and supervision rules by using the corresponding binary trees. The execution module is the interface between the robot and the control system. It translates the high level orders of the plan into low level orders comprehensible by the remotely operated machine. The execution module supports various protocols of communication (serial connection, Ethernet, FDDI).

5. SOLUTIONS FOR THE DEPENDABILITY OF ROBOTIC APPLICATIONS WITH PILOT

The solutions for dependability in the context of PILOT can be divided into two categories: features directly integrated in the language or implemented in its control system (internal mechanisms) and solutions related to the software development process (formal specification and verification, testing).

5.1. Internal mechanisms

The actions of PILOT comprise **precondition and supervision rules**. An action can be carried out only if its precondition is true. In the same way, when a supervision rule is satisfied, the associated processing is carried out. PILOT **plans can be modified during their execution**.

The environment of control of PILOT was equipped with a mechanism of **syntax-oriented edition** making it possible to guarantee the syntactic validity of the plan at each phase of its construction (insertion, modification, suppression of primitives) **[17]**. A **controller based on the formalism of "controllers' synthesis"** was also integrated into the control system.

5.2. Mechanisms related to the software development process

Software testing

Static and dynamic testing was applied to the interpreter which is one of the most critical modules of the control system of PILOT [18]. A simple simulator of robot was built for the dynamic testing operations in order to avoid damage to the real robot and its environment. The static test consisted, on the one hand, in reading the source code with the aim of detecting the programming errors and, on the other hand, in analyzing it in comparison with the algorithms of interpretation and the semantics of PILOT. As regards the dynamic testing, it consisted in defining testing data and applying them to the binary code of the interpreter. The testing data were defined by combining a functional approach with the experience feedback of the tests already carried out. For the definition of the representative sample of testing data, we adopted an incremental approach. The empty sequence was initially tested, and then the other primitives of the language were tested individually. Combinations in length, in width and in depth of the primitives of the language were then considered by increasing respectively, the number of elements in the sequences of the plan, the number of branches in parallel and conditional primitives, and the level of overlap. In order to have a limited set of testing data, we emitted stability assumptions: actions of the same type are interchangeable; the set of sequences resulting from the combination of any pairs of primitives is representative of the set of sequences comprising two or more primitives except for the problems of memory, etc.

Modelling, simulation and static verifications

The various modules of the control system software of the language PILOT were modelled using finite state automata and interpretation algorithms were defined for the various primitives of the language [6], but the automata were not checked.

The static and dynamic testing described above do not make it possible to guarantee the conformity of the interpretation of the plans to the operational semantics of the language PILOT. Complementary work was carried out to alleviate this disadvantage [19]. It consisted in modelling the algorithms of interpretation and checking their conformity in comparison with the operational semantics of the language in order to correct the possible dysfunctions and to regenerate the code of the interpreter starting from the validated model. Coloured Petri nets (CPN) were used for modelling and verification. The software used was Design/CPN which is currently replaced by CPN TOOLS (http://www.daimi.aau.dk/DesignCPN). Petri nets and more particularly CPN were selected for various reasons. Their graphic nature offers the user-friendliness necessary for a later use of the model as a means of communication between the various people implied in the development of the control system. They make it possible to represent relatively simply the various concepts of algorithmic and programming. The availability of tools, for the simulation and the checking of the models, was also an important criterion.

Test plans were generated using the approach adopted during the dynamic testing of the interpretation algorithms and their execution was simulated. The simulation making it possible to explore, in practice, only part of the execution paths, complementary work was carried out. From the CPN modelling the algorithms of interpretation and a test plan, the graph of accessible markings corresponding to the possible execution paths was generated using the Design/CPN tool. The graph of markings and the test plan were then transmitted to the verification program which examined, for each execution path, the satisfaction of the operational semantics of the language.

As far as the syntax-oriented edition mechanism mentioned above (subsection 5.1) is concerned, its conformity in comparison with a formal plan synthesizer was checked. It consisted in modelling PILOT plans and the edition operations in Prolog and then checking that all and only all syntactically valid plans were built [17]. This work was done using the SWI-Prolog tool.

In addition to the above verifications, a solution based on real time scheduling has been proposed in order to ensure real time properties of missions [12, 20]. It consists in modelling the whole control system of PILOT in terms of tasks, buffers and communication channels. From this model and from measurements performed on the platform, the maximum bounds on the occupation and the waiting time of its buffers are calculated [30] as well as the worst case response times of tasks [10]. Once response times of the control system' tasks are computed, one can obtain the response times of the actions and control structures of the language PILOT.

After this presentation of the solutions proposed for the dependability of robotics applications with PILOT, the next section addresses the results emanating from their application to PILOT and to its control system software.

6. *Results*

Precondition and supervision rules avoid unsafe execution of actions. If we consider for example the action "move forward" for a mobile robot equipped with detectors of obstacles, a rule of precondition could be the test of absence of obstacle. One of the supervision rules would be for example the test of presence of obstacles.

Thanks to the possibility provided by PILOT for runtime modification of plans, the operator can, in the event of dysfunction in the execution of a plan, make **modifications allowing the system to return in a satisfactory operating condition** (continuation of the mission or stop in a safe state). The **syntax-oriented edition mechanism** and the **controller** make it possible to preserve the advantages of the possibility of modifying plans dynamically. Indeed, the interpreted nature of the language and the possibility of modifying plans in the course of execution allow the execution of incomplete plans. One can thus launch the execution of a plan with missing parts such as the end of the main sequence or containing a parallel structure whose execution cannot finish because it is incomplete. The syntax-oriented edition avoids such situations. The controller ensures the semantic validity of the plan at runtime by disallowing modifications such as suppression of running actions, and allowing insertion of primitives only if it makes sense.

The **static and dynamic testing** made it possible to detect and correct errors of various types: error in the management of software interruptions and in the management of continuous actions termination, etc.

The **finite state automata** modelling the modules of the control system software and the **interpreter algorithms** provide a good basis for the prevention of errors (the application of formal verification methods is possible). Moreover, they make it possible to avoid errors due to the distortion of information throughout the design and implementation process of the control system software.

The **modelling of the interpretation algorithms with Petri nets** made it possible to note that simplifications were possible both on the level of the internal representation of a plan, and on the level of the algorithms of interpretation, and to apply these to the control system (suppression of a component in the internal representation of the plan, merging of the two sequence interpretation algorithms, etc.). The **simulation of the Petri net**s made it possible to detect termination problems (deadlock in the execution of an empty preemption primitive, etc.).

The **checking of conformity of the syntax-oriented mechanism** in comparison with a formal plan synthesizer allowed the validation of the proposed mechanism.

The work based on **real time scheduling** made it possible to characterize temporally the control system and to validate temporally PILOT plans before their execution.

7. *Discussion*

The supervising rules mechanism is equivalent to the exceptions mechanism and constitutes a solution for the implementation of fault tolerance.

As far as formal methods are concerned, the solutions proposed for the LAAS Architecture as well as for the PROCOSA system of the ONERA are mainly oriented towards the generation of controllers whereas the ORCCAD architecture mainly deals with static verification of missions. Both approaches have been experienced in the context of PILOT. These different studies show that although significant effort is needed for the adaptation of formal methods to the field of robotics, it is not only possible but very useful. Controller synthesis is particularly important for architectures such as PILOT which allow runtime modification of missions (the set of authorized missions is restricted to those satisfying some formally defined safety constraints).

Virtually no experience on the use of software engineering techniques for the design of robot control architectures has been reported. Our experimentation of software engineering techniques of modelling, simulation and testing with the PILOT architecture shows that some specificities of robotics need to be taken into account, in particular for the testing (events generated by the robot which are not easily controllable, simulator required for hostile environments or to avoid damage to the robot under test or the environment, …). The characteristics of the mission programming (existence of elementary actions which impacts the structuring of plans) approach can also be exploited for the definition of testing data set.

Operator intervention on the course of the mission is a requirement of robotics applications. It is part of the features needed for fault tolerance of robotics applications. Making it possible to dynamically modify missions is therefore an asset for dependability of such applications. Nevertheless in order to be fully beneficial, such possibility needs to be complemented with additional features such as a syntax and semantic-oriented edition of missions. At first glance, such a mechanism seems time-consuming, but this is not necessarily true. Indeed, in our case study with PILOT, the verification needed in case of modification is limited to the direct enclosing control structure.

8. *Conclusion and future work*

Formal methods have been used only in a few robotic architectures. In the same way, fault tolerance mechanisms are exploited little. One can also note the small number of experiments reported as regards rigorous testing in the realization of robotic architectures. Structured testing is however very important in the process of development of dependable software. Globally, a significant effort is needed in the application of software dependability techniques to the design and implementation of robotic architectures. The use of distribution in robotic architectures provides the opportunity of improving fault tolerance through mechanisms such as replication. The work completed within the framework of the language PILOT and its control system brought generic solutions for the dependability of robotics applications: a precondition rule which ensures the safety of action execution, monitoring rules which make it possible to recover from an action failure, a syntax-oriented edition mechanism which assists the operator in the design of plans and reduces the risks of faults, possibility to modify missions during execution which makes it possible to recover from errors, mechanism of security of modifications in the course of execution. The static and dynamic testing as well as the formal modelling and checking applied to the interpreter of plans can also apply to other mission programming environments. Several perspectives can be envisioned in the continuation of the former work. One of them is the automation of the testing process. It is necessary to generate proper traces and to design and implement traces analysis methods for fault detection. Another perspective is the use of theorem proving to extend the validation of the PILOT syntax-oriented edition approach which was limited to plans of a maximum size of 14 (maximum number of elements of the list modelling the plan) due to the combinatory explosion of the model-checking approach formerly used.

REFERENCES

[1] Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C., “Basic concepts and Taxonomy of Dependable and Secure Computing”, IEEE Trans. On Dependable and Secure Computing, January-March 2004, 1(1):11-33.

[2] Barbier, M., Gabard, J. F., Vizcaino, D., Bonnet-Torrès, O., “ProCoSA: a software package for autonomous systems supervision”, 1st National Workshop on Control Architectures of Robots: software approaches and issues, Montpellier, France, April 2006.

[3] Barrouil, C., Lemaire, J., “Advanced Real-Time Mission Management for an AUV”, SCI NATO RESTRICTED Symposium on Advanced Mission Management and System Integration Technologies for improved Tactical Operations, Florence, Italy, Sept. 1999.

[4] Bernard, B. E.,Gamble, E. B., Rouquette, N. F., Smith, B., Tung, Y. W., Muscettola, N., Dorias, G. A., Kanefsky, B., Kurien, J., Millar, W., Nayal, P., Rajan, K., Taylor, W., “Remote agent experiment DS1 Technology Validation Report”, Ames Research Center and JPL, 2000.

[5] Brooks, R. A., “A robust layered control system for a mobile robot”, IEEE Journal of Robotics and Automation, March 1986, pages 14-23.

[6] Fleureau, J.L., "Vers une méthodologie d'un système de programmation de télérobotique: comparaison des approches PILOT et GRAFCET", PhD thesis, Université de Rennes 1, France, 1998.

[7] Goldman, R.P., Musliner, D.J., Pelican, M.J., “Using Model Checking to Plan Hard Real-Time Controllers”, In Proceedings of the AIPS2k Workshop on Model-Theoretic Approaches to Planning, Breckenridge, Colorado, April 14, 2000.

[8] Hassoun, M., Laugier, C., “Reactive motion planning for an intelligent vehicle”, In Intelligent Vehicles'92 Symposium, Detroit, July 1992, pp. 259-264.

[9] Hayes-Roth, B., “A blackboard architecture for control, Artificial Intelligence”, 1985, 26:pp. 251-321.

[10] Joseph, M., Pandia, P., “Finding Response Time in a Real- Time System”, Computer Journal, 1986, 29(5):390-395.

[11] Kapellos, K., Simon, D., Espiau, B., “Control laws, tasks, procedures with ORCCAD; application to the control of an underwater arm”, In 6th IARP (International Advanced Robotic Program), La Seyne sur Mer, France, 1996.

[12] Legrand, J., « Contribution à l’ordonnancement de tâches partagant des tampons », PhD thesis, University of Brest, December 2004.

[13] Lussier, B., Lampe, A., Chatila, R., Guiochet, J., Ingrand F., Killijian, M.-O., Powell D., “Fault Tolerance in Autonomous Systems: How and How Much?” 4th IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, Nagoya, Japan, June 16-18, 2005.

[14] Le Rest, E., « PILOT: un langage pour la télérobotique », PhD thesis, Université de Rennes 1, France, 1996.

[15] Lumia, R., Fiala J., Wavering, A., “The NASREM robot control system and testbed”, IEEE Journal of Robotics and Automation, 1990, 5(1), pp. 20-26.

[16] Nana, L., Marcé, L., Opderbecke, J., Perrier, M., Rigaud, V., “Investigation of safety mechanisms for oceanographic AUV missions programming”, In Proceedings of the IEEE OCEANS’05 Europe Conference, Brest, June 2005.

[17] Nana Tchamnda, L., Marcé, L., “Vers une programmation sûre avec PILOT”, MSR’2001, Colloque Francophone sur la Modélisation des Systèmes Réactifs, Toulouse, France, October 2001.

[18] Nana Tchamnda, L., Nicolas, V-A., Marcé, L., “Towards a formal approach for the regeneration of PILOT control system”, 6th World Multiconference on Systemics, Cybernetics and Informatics, SCI’2002, IEEE Venezuela, Orlando, Florida, July 2002.

[19] Nana, L., Legrand J., Singhoff, F., Marcé, L., “Modelling and Testing of PILOT Plans Interpretation Algorithms”, Multi-conference on Computational Engineering in Systems Applications, CESA'03, IEEE, Lille, July 2003.

[20] Nana, L., Singhoff, F., Legrand, J., Vareille, J., Le Parc, P., Monin, F., Massé, D., Marcé, L., Opderbecke, J., Perrier, M., Rigaud, V., “embedded intelligent supervision and piloting for oceanic AUV”, In Proceedings of the IEEE OCEANS’05 Europe Conference, Brest, June 2005.

[21] Nilsson, N., “A mobile automation: an application of artificial intelligence techniques”, In Proc. Int. Joint Conf. on Artificial Intelligence, 1969, pp. 509-520.

[22] Nilsson, N., “Shakey the robot”, Technical Report 323, SRI, Menlo Park, CA.

[23] Perrier, M., Kalwa, J., “Intelligent Diagnosis for Autonomous Underwater Vehicles using a Neuro-Symbolic System in a Distributed Architecture”, In Proceedings of the IEEE OCEANS’05 Europe Conference, Brest, June 2005.

[24] Py, F., Ingrand, F., “Dependable Execution Control for Autonomous Robot”, IROS 2004 (IEEE/RSJ International Conference on Intelligent Robots and Systems), Sendai, Japan, September 28 - October 2, 2004.

[25] Ramadge, P. J., Wonham, W. M., “The control of discrete event systems”, Proceedings of the IEEE, Special issue on dynamics of discrete event systems, 1989, vol. 77, no. 1, pp. 81-98.

[26] Rosenblatt, J., “DAMN: A distributed architecture for mobile navigation”, Journal of Experimental and Theoretical Artificial Intelligence, 9(2/3), 1997, pp. 339-360.

[27] Rutten, E., “A framework for using discrete control synthesis in safe robotic programming”, Rapport de recherche, INRIA, 2000.

[28] Schneider, S., Chen, V., Pardo-Castellote, G., Wang, H., “ControlShell: A software architecture for complex electro-mechanical systems”, International Journal of Robotics Research, Special issue on Integrated Architectures for Robot Control and Programming, 1998.

[29] Seabra Lopes, L., Camarinha-Matos, L.M., “Learning to diagnose failures of assembly tasks”, Annual Review in Automatic Programming, 1994, vol. 19, pp. 97-103.

[30] Singhoff F., Legrand, J., Nana, L., Marcé, L., “Extending Rate Monotonic Analysis when Tasks Share Buffers”, In the DAta Systems in Aerospace conference (DASIA 2004), Nice, July 2004.

[31] Tigli, J.Y., « Vers une architecture de contrôle pour robot mobile orientée comportement, SMACH », Thèse de Doctorat, Université de Nice - Sophia Antipolis, Janvier 1996.

[32] Tomatis, N., Terrien, G., Piguet, R., Burnier, D., Bouabdallah, S., Arras, K. O., Siegwart, R., “Designing a secure and Robust Mobile Interacting Robot for the Long Term”, In Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, September 14-19, 2003, pp. 4246-4251.

[33] Turro, N., « MaestRo: Une approche formelle pour la programmation d'applications robotiques », Thèse de doctorat, Université de Nice, Sophia Antipolis, Septembre 1999.