Embedded intelligent supervision and piloting for oceanographic AUV

Conference Paper · July 2005

DOI: 10.1109/OCEANSE.2005.1513174 · Source: IEEE Xplore

CITATIONS
11

READS
88

11 authors, including:

Laurent Tchamnda Nana
Université de Bretagne Occidentale
101 PUBLICATIONS **926** CITATIONS
SEE PROFILE

Frank Singhoff
Université de Bretagne Occidentale
119 PUBLICATIONS **1,069** CITATIONS
SEE PROFILE

Jérôme Legrand
Ellidiss Technologies
23 PUBLICATIONS **590** CITATIONS
SEE PROFILE

Jean Vareille
Université de Bretagne Occidentale
42 PUBLICATIONS **102** CITATIONS
SEE PROFILE

Embedded intelligent supervision and piloting for oceanographic AUV

L. Nana, F. Singhoff, J. Legrand, J. Vareille, P. Le Parc, F. Monin, D. Massé and L. Marcé

Computer Science Department, University of Brest

20, Avenue Le Gorgeu

CS 93837 - 29238 Brest cedex 3 - FRANCE

{nana,singhoff,jlegrand,vareille,leparc,monin,masse,marce}@univ-brest.fr

J. Opderbecke, M. Perrier and V. Rigaud

IFREMER, Underwater System Department

ZP de Bregaillon

83507 La Seyne sur Mer -FRANCE

{Jan.Opderbecke,Michel.Perrier,Vincent.Rigaud}@ifremer.fr

***Abstract* -The main goal of the work presented in this paper is to associate a basic AUV (Autonomous Underwater Vehicle) control system such as ACE (an IFREMER control system) with a high level command part. This high level part is provided with complementary features needed by the initial AUV subsystem namely, a language allowing plans modification during their execution and a diagnosis and analysis system. The latter checks continuously the information provided by the sensors in relation to some predefined rules in order to ensure the good working of the AUV within its environment.**

I. INTRODUCTION

Most of the AUV control systems are adapted to simple preprogrammed missions such as following roads composed of a series of waypoints. The «plans» of these missions are loaded at the beginning but they are not modifiable during execution. The management of the incidents and of unforeseen situations is limited to the stopping of the mission with different levels of seriousness. It doesn't allow the re-programming of the mission in reaction to the occurred events. This mechanism is well suited for serious faults (essential sensors, propulsion, waterway, etc.). For less serious dysfunctions (gap of the road, unforeseen submarine flowing, undervalued energy consumption, «small» obstacles), such a stop is not often the proper solution, since it is difficult to envision to restart immediately the whole procedures which are necessary for the resumption of the mission. Once immersed, the communication capacity of the AUV is most often reduced to an acoustic low data-rate connection, unsuitable with the transmission of the whole information collected by the sensors.

In order to overcome the problems above, we propose, on the one hand, to associate the current system with a high level command part provided with a mission programming language allowing plans modification during their execution and comprising a diagnosis and analysis system checking continuously the information provided by the sensors in relation to some predefined rules which should be satisfied in order to ensure the good working of the AUV within its environment. On the other hand, we propose to associate the control system with a system for communication monotoring.

In the second section of this paper, a brief overview of missions programming languages is given.

The language PILOT and its control system [14], suggested for the high level command part, are presented in the third section. It makes it possible to modify plans during their execution. The control system of PILOT is modular and comprises a basic inference engine.

The fourth section is dedicated to various solutions for AUV missions' safety and supervision. The first approach is based on real-time scheduling. The goal is to ensure temporal properties of plans. The second approach is related to the monitoring of communication failures. In the third approach, a system named ADVOCATE II, providing mechanisms for a deepened diagnosis in case of failure, is considered. IFREMER Toulon has proposed this system in the context of a European project [16], which led to modular, generic and opened software architecture for the control of autonomous robotic systems and their recovery in case of dysfunction. It can be associated with the inference engine of the control system of PILOT or other similar control systems, in order to increase the safety of AUV applications. This section ends with the verification of safety properties on PILOT control system using proof approaches.

The paper ends by a conclusion in the fifth section.

II. OVERVIEW OF MISSIONS PROGRAMMMING LANGUAGES

Missions programming languages are based on the existence of elementary actions provided by a subsystem to allow the specification of the robotics applications in term of scheduling of the elementary actions. To date, three techniques were developed in the design of missions programming languages: extension of general purpose languages (as C, Ada or LISP) with robot-like directed libraries, the creation of languages specific to the field of robotics, and the modification of languages of control such as LUSTRE and SIGNAL.

The disadvantage of the first approach is the inadequacy from the point of view of the specification and the determinism of the execution. The second approach is of multiple interests. The languages are closer to specification languages, capture the semantics of the field better and produce consequently clearer and more concise programs. Many languages dedicated to the programming of manufacturing robotics applications were thus created at the end of the seventies: LM [13], VAL [19], etc. Their capacity of expression however is very directed towards the control of arm manipulators, which compromises their extensibility to a wider domain such as that of mobile robotics. In the same category, other languages, generic insofar as they are not dedicated to only one type of robot, were created in the world of research. It is for example the case of the language of handling of the robotics actions and the intermediate goals used by the C-PRS [6] subsystem of the decisional level of the robotics architecture of the LAAS. However, these languages do not take into account, in their semantics, the kinematics and dynamic aspects specific to robotics such as the sequence of trajectories. Control languages have an expressivity much nearer to the programming of robotics missions than that of general purpose languages. They have a rigorously established semantics (operational semantics) and tools of simulation and/or checking and/or analysis, which represent an advantage for the safety of the robotics applications. However, their use in a robotics context requires adaptations.

From study of the robotics missions programming languages of robot-like missions, it arises clearly that none of the evoked approaches constitutes alone the ideal solution. Such a solution, although dependent on architecture and on the target application, can emerge from a combination of approaches among the preceding ones. Such a language would have:

- To allow the application of formal methods of checking and analysis: this passes by the availability of a clearly defined semantics and the availability of tools of simulation, checking and analysis of properties.
- To have a sufficient expressivity: it must provide the structures of basic control, which one finds in the traditional imperative languages such as PASCAL or C, but also structures for the parallel programming, and structures adapted to the reactive programming (management of events and/or signals, …).
- To allow intuitive programming of the control of the applications, for example, by the use of a graphic formalism.
- Make it possible to program at robot-task level.
- Provide a good compromise between extensibility and specialization.
- To integrate possibilities of operator action.
- To allow on line modification of the program if its execution does not proceed as envisaged.

The last two points are particularly useful for fault-tolerance which is an essential complement to the methods of formal checking and analysis to fully ensure the reliability of the applications. After this overview of missions programming languages, the next section deals with the language PILOT and its control system, suggested for the high level command part to associate with an AUV control subsystem such as ACE [17].

III. PILOT : A LANGAGE AND A CONTROL SYSTEM

A. The language PILOT

The language PILOT (Programming and Interpreted Language of actiOns for Telerobotics) is a graphical and interpreted language dedicated to the remote control of systems. It has been designed and built by the LISYC laboratory shared by UBO (Université de Bretagne Occidentale) and ENIB (Ecole Nationale des Ingénieurs de Brest), in order to easy and to secure the programming of missions [11] [3] [15]. It is based on the notion of action. Two kinds of actions are defined (see figure 1): elementary actions which have their own end and which generally end when their predefined goal is reached, and continuous actions whose end is triggered by another primitive of the language.

Fig. 1 Elementary and continuous actions

The language PILOT provides different control structures for plans building:

- Sequentiality: it is made of a succession, possibly empty, of actions and/or control structures. Figure 2 shows an example of sequence made of two elementary actions (Action1 and Action2). The execution of Action2 starts after the end of Action1.

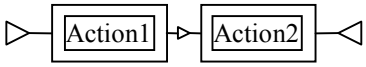

Fig. 2. Example of sequence

- Conditional: It is made of one or more alternatives ordered from top to bottom. Each alternative comprises a condition followed by a sequence. The only sequence executed is the first one whose condition is true. Figure 3 shows an example of conditional.

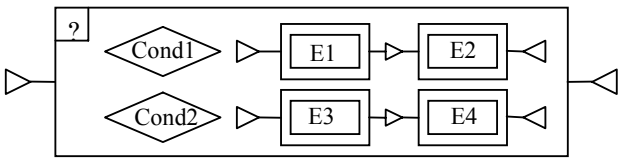

Fig. 3. Example of conditional

- Iteration: it comprises a continuation criterion followed by a sequence. The criterion is either a number of loops (fix iteration) or a Boolean expression (conditional iteration). Figure 4 and 5 show examples of fix and conditional iteration.

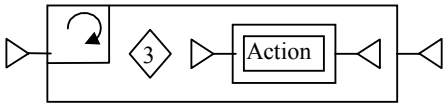

Fig. 4. Example of fix iteration

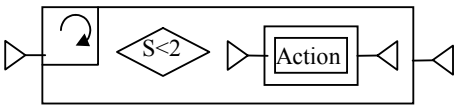

Fig. 5. Example of conditional iteration

- Parallelism: it is made of sequences which execute in parallel. Its execution ends when all the sequences reach their end. An example of parallelism is shown in figure 6.

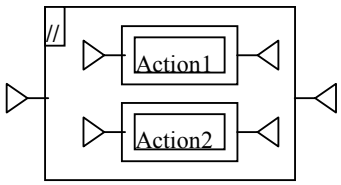

Fig. 6. Example of parallelism

- Preemption: as parallelism, it is made of sequences which execute in parallel, but unlike the latter, when one of the sequences ends, it triggers the end of the other sequences and the end of the preemption. We illustrate the use of preemption in figure 7.

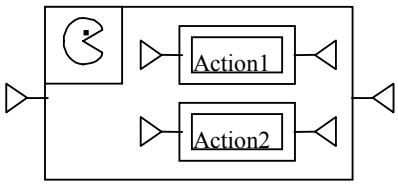

Fig. 7. Example of preemption

B. The control system

The control system is the interface between the user and the remotely operated machine. It comprises six concurrent modules (figure 8):

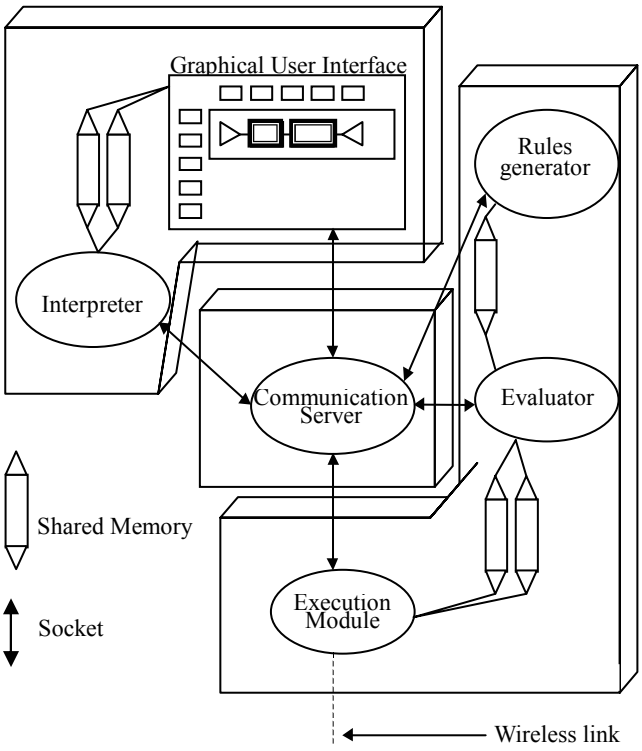

Fig. 8. PILOT control system

- a graphical interface also called man-machine interface (MMI),
- an interpreter,
- a communication server,
- a rules generator,
- an evaluator and
- an execution module or driver.

These processes communicate through sockets and shared memory and can execute either on a single computer or on a network of computers.

The man-machine interface provides different features for designing plans. It stores the plan into a memory space shared with the interpreter. The interpreter reads the plan from the shared memory and sends orders (precondition request, order to start an action, ...) to the other modules in order to achieve the plan execution. The communication server handles inter-process communications. The purpose of the rules generator is to transform character strings of precondition and supervising rules into binary trees. It stores the result into shared memory for future use by the evaluator. The rules evaluator is in charge of calculating the Boolean expressions of precondition and supervising rules. The execution module is the interface between the robot and the control system. It translates high level orders of the plan into low-level orders which are understandable by the remotely operated machine. The execution module is the only one comprising a part which is specific to the remotely operated robot. This makes the control system of PILOT easily adaptable to robots of different natures.

In the next section, we propose some solutions for AUV missions safety. The first one is based on real-time scheduling and aims at ensuring that real-time constraints of the application will be met. The second consists in monitoring communication failures. The third solution is an artificial intelligence based approach for a deepened diagnosis an recovery of failures and the last one uses proof approaches.

IV. MISSIONS SAFETY AND SUPERVISION

A. A safety approach based on real-time scheduling

The goal is to ensure temporal properties of plans [8]. The control system of PILOT as well as its execution support part embedded on the robot can be modelled in terms of tasks, buffers and communication channels.

From this model and from measures performed on the platform, we calculate the maximum bounds on the occupation and the waiting time of its buffers [20] and worst case response time of tasks [7, 1]. The response time is the maximum delay between the time the task becomes ready to run and the time the task ends its job. Finally, we apply the holistic analysis to determine the response time of the control system of PILOT [21]. The holistic analysis method makes it possible to compute, for the worst case, the execution time of a set of tasks possibly located on different processors and which have to be executed according to some precedence constraints. When response times of PILOT control system tasks are computed, we can obtain the response time of the actions and control structures of the language PILOT. Thanks to these response times, we can finally validate temporally a plan built by an operator, before its transfer on the robot

and therefore before its execution.

1. How to compute task response times

Since 1980, to check or compute task response time of an application made of concurrent tasks, many models, methods and tools were proposed (e.g. Petri Nets, synchronous languages [9], …). One of them, usually called "Rate Monotonic Analysis" is part of a larger set of quantitative methods: the real time scheduling theory. This theory helps the system designer to predict the timing behaviour of a set of real time tasks with *scheduling simulation* and *feasibility tests*. Scheduling simulation requires, first to compute a scheduling on a given time interval and second, to look for timing properties in this computed scheduling. On the contrary, feasibility tests allow the designer to investigate a set of real time tasks without computing scheduling. The first real time scheduling theory contributions were proposed 30 years ago [12]. The theory was strongly extended to cope with many application requirements and was successfully used in many projects [2, 18].

In the most simple task model, each task periodically performs a treatment. This "periodic" task is defined by three parameters: its deadline (*Di*), its period (*Pi*) and its capacity (*Ci*). *Pi* is a fixed delay between two wake up times of the task *i*. Each time the task *i* is woken up, it has to do a job whose execution time is bounded by *Ci* units of time. This job has to be ended before *Di* units of time after the task wake up time.

As mentioned above, from a set of periodic tasks, two kinds of analysis can be performed: *scheduling simulation* and *feasibility tests*.

Scheduling simulation consists in predicting for each unit of time, the task to which the processor should be allocated.

Different kinds of *feasibility tests* exist: tests based on processor utilization factor, task response time designed to check task deadlines and tests based on buffer utilization factor designed to check buffer overflow.

The second feasibility test mentioned above consists in comparing the worst case response time of each task with its deadline. In [7, 1], Joseph, Pandia, Audsley *et al.* have shown that *ri*, the worst response time of a task *i*, can be computed as follows:

$$ri = \max_{q=0,1,2}(Ji + Bi + wi(q) + qPi) \quad (1.1)$$

Where

$$wi(q) = (q+1)Ci + \sum_{\forall j \in hp(i)} \left\lceil \frac{Jj + wi(q)}{Pj} \right\rceil$$

And

$$\forall q: wi(q) \le (q+1)Pi$$

And where *hp (i)* is the set of tasks which have a priority greater than *i*, *Bi* is a bound on shared resource blocking times and *Ji* is a bound on the task wake up time jitter.

Finally, Tindell had shown in [1] how equation 1.1 can be extended to compute worst case response times of tasks running on a distributed system.

2. From task response times to PILOT plans response times

When the response time of the control system of PILOT are computed, one can compute the worst case execution time of PILOT plans.

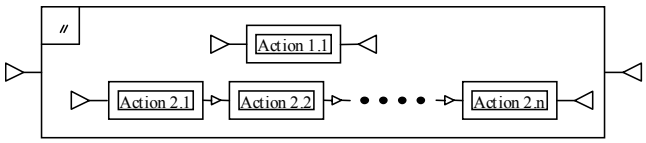

Fig. 9. An example of plan with parallelism

Figure 9 shows an example of plan composed of two sequences of actions: the sequence *Action1.1* and the sequence *Action2.1/Action2.2/.../Action2.n*. The two sequences are run in a parallel way. The worst case execution time of such plan can be computed as follows.

1. First, the worst case execution time of each sequence has to be evaluated. The sequence worst

   $$rs = \sum_{1 \le k \le n} Wk + ri$$

   case execution time can be evaluated by :

   Where *Wk* is the time required by the robot to perform the action *k* of the sequence, and *ri* is the response time of the PILOT control system.

2. Second, from the worst case response time of a sequence, one can evaluate the worst case execution

   $$rplan = \max_{\forall 1 \le j \le 2}(Sj)$$

   time of the plan of figure 3 with :
   Where *Sj* is the worst case execution time of the sequence *j* composing the parallel statement.

This method can be applied on the different plan statements provided by PILOT. In [8], one can find a detailed description of how this method can be applied to PILOT preemption, conditional and iteration statements.

B. Monitoring communication failures

The AUV can communicate when it is on the surface thanks to a hertzian antenna. It can also communicate by acoustic way when it is in diving. The band-width available on the surface can reach the Mbit per second with a sufficient quality of communication. On the other hand in diving the band-width is much smaller, some kilobits per second, the connection is much more unreliable because of the variations of the acoustic index of the sea water which depend on the temperature and the salinity and which sometimes place the AUV in zones of shade, or rain which can create sound waves which blurs the signal. The possibilities of communication in diving do not allow to teleoperate the AUV with reliability. Consequently it is necessary to equip the AUV with a decisional system enabling it to carry out choices similar to those which would be taken by a human operator.

The AUV thus has an autonomous behavior. Nevertheless, some operations remain critical like navigation in low depth and recovery at sea. For this reason it should be interesting to equip it with a system of supervision making it possible to supervise the critical phases and to enable the change of plans of mission. The language PILOT is particularly adapted to the dynamic change of the plans of mission. To be able to profit from this quality it is thus necessary to be able to communicate

during the mission and afterwards until the recovery of the vehicle. As the communication is not reliable in diving, but can also be disabled on the surface because of the movements of the sea and of the small size of the vehicle, we propose to take into account the quality of communication like additional parameter in entry of the decisional system. The measurement of the quality of communication can very simply be taken by the sending of very small time stamped frames by the vehicle immediately returned by the distant supervisor. This measurement of round trip time similar to an ICMP "ping" but different in the sense that it is carried out local decisional level up to the distant decisional level, makes it possible to quantify the latency time of communication. According to this one, the embedded decisional system will be able to estimate if the received orders of the remote supervision correspond to its current situation or to a too old situation so that they can be executed validly. We resume the studies carried out for the terrestrial systems [22, 10] by adapting them to the context of the AUV.

C. An Artificial Intelligence based approach for missions supervision : the system Advocate II

The principal objectives of ADVOCATE II [16] are to increase the performance of AUV and terrestrial robotics applications, to increase the safety of the system and its environment. The adopted approach consists in adding intelligence in existing or new architectures of control. The techniques of artificial intelligence used are: BBN (Bayesian Belief Network), fuzzy logic and neuro-symbolic systems.

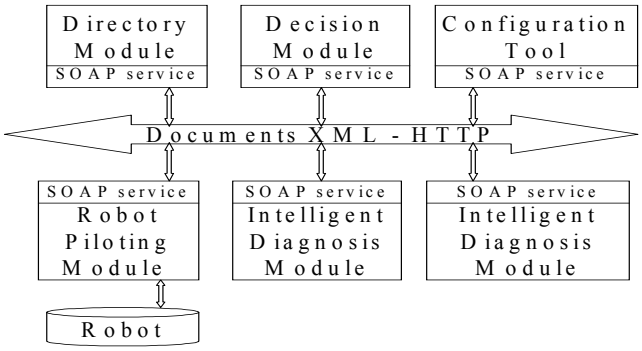

Fig. 10 Architecture of Advocate II

The modules of the Advocate II system (figure 10) are described below.

1. Robot piloting modules

Each *robot piloting module* runs on a specific robot and communicate directly with the sensors and effectors of the vehicle. It offers the following functionalities: management of the plans of missions, self-piloting, guidance and navigation. It interacts with the modules of Intelligent Diagnosis and Decision while providing: data (sensors, effectors, computation results, alarms) emanating from the Robot, as well as information relating to the mission and actions.

2. Directory module and configuration tool

The role of the *directory module* is the administration of the list of recorded intelligent modules and the administration of the list of the services offered by those modules. The role of the configuration tool is the positioning of the parameters of the system and the choice of the intelligent module adapted for the resolution of a particular problem.

3. The decision module

The *decision module* can be regarded as the central unit of the architecture. It interacts with:

- The *GUI*, if it exists, by transmission of information intended for the operator.
- *Robot Piloting Modules* of the vehicle for the setting of recovery plans produced like result of a process of diagnosis and recovery.
- *Intelligent Diagnosis modules*: request/reception of the diagnoses, request/reception of actions of compensation. These modules need to have a good knowledge on the whole of the system.

4. Intelligent Diagnosis module

The main goal of the *Intelligent Diagnosis Module* is the resolution of the problems of diagnosis and the recovery actions. It comprises two parts: a generic part and a part specific to the problem. The specific part is formed of a knowledge base (network of neurons, rules base or probabilistic networks). Its role is to carry out actions of diagnosis and/or recovery and to maintain a "log book" of pre/post - treatment. The generic part provides as for it the interface and the functionalities common to all the intelligent modules (method of solution, inference engine, etc). The modules of intelligent diagnosis have two operating modes: an "on line" operating mode and an "off line" operating mode. In the off line mode, they make it possible to build, update or to revise the Knowledge Base to be used on line. In this mode, their functionality is dependent on the technique of Artificial Intelligence (AI) used [16]. In on-line mode they use the Base of Knowledge to treat the requests in order to produce the diagnoses and actions of recovery.

D. Formal Approaches

The formalization of the semantics of PILOT enables us to work on other formal approaches to further ensure the safety of generated plans, e.g. using a proof checker like the PVS system.

The choice of PVS is in part due to our experience in using this tool for the formal checking of protocols (see [4, 5]).

A preliminary work is to express the logical rules from which a plan can be built in the language of PVS, that is a higher-order typed logic devoted to writing formal specifications and checking formal proofs, to show that the process of constructing plans meet consistency (and completeness) requirements. Hence from all generated plan we can derive an exact formal semantics, which allows us to use theorem proving or static analysis techniques to prove the satisfaction of specifications.

V. CONCLUSION

In this paper, we proposed solutions allowing meeting the needs as regard capacity of modification of missions in the course of execution. Such capacity is non-existent in the majority of the traditional AUV control systems. To be entirely effective, such a capacity requires to be supplemented by complementary safety mechanisms, in order to avoid disastrous consequences of a bad modification of the plan of mission. The approach suggested consists in associating with the system of control of the AUV, the control system of PILOT. This last makes it possible to modify missions in the course of execution and is, in addition, equipped with supplies making it possible to make safe the modifications and to assist the operator in the development and the modification of the plan of mission (edition directed by syntax, protocol of modification in the course of execution taking of account the semantics of the mission). We also proposed solutions allowing meeting the needs as regard supervision and checking of plans of missions of AUV. Two approaches were proposed for the supervision of AUV applications. The first is based on the results of work undertaken by the IFREMER within the framework of European project ADVOCATE II. It is based on the use of techniques of artificial intelligence to carry out a thorough diagnosis of the failures and to allow the recovery of error. The second consists in defining levels of quality of connections of communication between the control unit on the surface and the AUV in order to detect the problems of communication and to make it possible to react in a way appropriate with the fall of quality or the rupture of the communication. Two approaches were proposed for the checking in the conceptual phase of the mission. The first one is based on the use of techniques of real time scheduling to check the respect of temporal constraints on a plan of mission. The second one relies as for it on the use of the techniques of proofs to check the respect of properties within mission or its environment.

One of the prospects for the work presented in this article is the implementation of the solutions suggested. We also consider, in collaboration with TNI-Valyosis, to integrate into tool STOOD and our tool for checking of properties of tasks (Cheddar), a translator towards AADL of their respective models of representation, in order to allow them to inter-operate. The interest is to be able, starting from the Cheddar model, to generate a description usable by the various tools of checking and in particular those associated to STOOD, and conversely of being able, starting from models resulting from STOOD, to generate models usable by the Cheddar tool. The implementation of the application of the evoked formal techniques of proof is also in hand. It initially aims at validating the approach of construction directed by the syntax of PILOT plans.

REFERENCES

[1] Audsley A.N., A.Burns, M.Richardson, and K.Tindell. “Applying new scheduling theory to static priority pre-emptive scheduling”. Software Engineering Journal, pages 284-292, 1993.

[2] Cottet F., J. Delacroix, C.Kaiser, and Z.Mammeri. “Scheduling in Real Time Systems”. John Wiley and Sons Ltd editors, 2002.

[3] Fleureau J.-L., « Vers une méthodologie de programmation d'un système de télérobotique : comparaison des approches PILOT et Grafcet », PhD thesis, University of Rennes 1, July 1998.

[4] Groote J.F., F. Monin and J. Springintveld. “A computer checked algebraic verification of a distributed summation algorithm,” Formal Aspects of Computing. Springer Verlag. Published Online, 2004

[5] Groote J.F., F. Monin and J.C. Van de Pol. “Checking verifications of protocols and distributed systems by computer”, Proceedings of Concur'98, Sophia Antipolis, LNCS 1466, pp. 629-655, Springer Verlag, 1998.

[6] Ingrand F. F., R. Chatila, R. Alami and F. Robert. “PRS: a high level supervision and control language for autonomous mobile robots”. In proceedings of the IEEE International Conference on Robotics and Automation, Volume 1, pp. 43-49, Minneapolis , USA, 1996.

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

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

[9] Le Guernic P., T. Gautier, M. L. Borgne and C. L. Maire. “Programming Real-Time application with SIGNAL”, INRIA Rennes, Technical Report N°1446, 1991.

[10] Le Parc P., J. Vareille et L. Marcé. “Web remote control of machine-tools: the whole word within less than one half second” *ISR 2004: International Symposium on Robotics 2004* - Paris (France), March 2004

[11] Le Rest E. « PILOT : un langage pour la télérobotique », PhD thesis, University of Rennes 1, June 1996.

[12] Liu C.L. and J.W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment”. Journal of the Association for Computing Machinery, 20(1):46--61, January 1973.

[13] Mazer E. and J. Miribel. “Le langage LM : Manuel de référence », Cépaduès éditions, 1984.

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

[15] Nana Tchamnda L., Marcé L. « Vers une programmation sûre avec PILOT », MSR’2001, Colloque Francophone sur la Modélisation des Systèmes Réactifs, Toulouse, France, 2001.

[16] Perrier M. “ADVOCATE - Data collection report”, Rapport Interne, IFREMER, DNIS/SM/RNV/00.032, 2000.

[17] Rigaud V., Michel J. L., Ferguson J.S., Laframboise J.M., Crees T., Leon P., Opderbecke J., Chardard Y., « First Steps in IFREMER’s Autonomous Underwater Vehicle Program – A 3000m Depth Operational Survey AUV for Environmental Monitoring », ISOPE 2004, vol. 2. pp 203-208, Toulon, France, Mai 2004.

[18] SEI. “The Rate Monotonic Analysis”. Technical report, In the Software Technology Roadmap. http://www.sei.cmu.edu/str/

descriptions/rma_body.html, September 2003.
[19] Shimano B., C. Geschke and C. Spalding, “VAL-II : A robot programming language and control system”, In Proceedings of the First International Symposium on Robotics Research, pp. 917-940, Bretton Woods, 1983.
[20] Singhoff F., J. Legrand, L. Nana, and L. Marcé. “Extending Rate Monotonic Analysis when Tasks Share Buffers”. In the DAta Systems in Aerospace conference (DASIA 2004), Nice, July 2004.
[21] Tindell K.W. and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming, 40(2-3):117--134, April 1994.
[22] Vareille J., P. Le Parc et L. Marcé. “Web remote control of mechanical systems: delay problems and experimental measurements of Round Trip Time”, 2nd Workshop CNRS-NSF Applications of Time-Delay systems, Nantes, September 2004.
