

ProCoSA: a software package for autonomous system supervision

Magali BARBIER¹ – Jean-François GABARD¹
Dominique VIZCAINO² – Olivier BONNET-TORRÈS^{1,2}

¹ ONERA/DCSD – 2 av. Edouard Belin – 31055 Toulouse cedex 4 - FRANCE
{ Magali.Barbier, Jean-Francois.Gabard, Olivier.Bonnet }@onera.fr

² SUPAERO/LIA – 10 av. Edouard Belin – 31055 Toulouse cedex 4 - FRANCE
Dominique.Vizcaino@supaero.fr

Abstract

Autonomy is required onboard uninhabited vehicles that move in partially known and dynamic environments. This autonomy is made possible thanks to the use of embedded decisional software architectures. This paper presents ProCoSA, an asynchronous Petri net-based tool that enables implementing such architectures. It allows procedure programming and execution in autonomous systems. Vehicle behaviours are modelled using Petri nets; a Petri net player runs the model and manages links with other software components. Several decisional architectures developed using ProCoSA for different types of vehicles - Autonomous Underwater Vehicle (AUV), autonomous Uninhabited Aerial Vehicle (UAV) and autonomous Uninhabited Ground Vehicles (UGVs) - are described in this paper, together with associated experiments and results.

1. Introduction

Research on autonomy is performed for Uninhabited Ground Vehicles (UGVs), Uninhabited Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs) and space vehicles. Autonomy is characterised by the level of interaction between the vehicle and a human operator: the higher level the operator's decisions are, the more autonomous the vehicle is. Between tele-operation (no autonomy) and full autonomy (no operator intervention), there are several ways to allow a vehicle to control its own behaviour during the mission [1].

Autonomous vehicles that move in partially known and dynamic environments have to deal with asynchronous disruptive events. Hence the need for implementing onboard decision capabilities that allow the vehicle to perform the mission even when the initial plan prepared offline is not valid any more. Decision capabilities, which guarantee the adaptability of the vehicle to variable environmental conditions, must be implemented in a dedicated architecture able to manage the components of

the whole control loop {perception, situation assessment, decision, and action}. The capacity to integrate environmental information given by sensors and to evaluate the current state is indeed essential for the vehicle to assure its own safety and the desired level of autonomy.

In an embedded decisional software architecture, a high level function is required to control mission execution: it supervises the nominal execution and triggers reactions to disruptive events. This function includes interactions with the physical system through dedicated control algorithms and deliberative task management.

The supervision function considered in this paper, sometimes called mission execution control function, does not include offline mission preparation, task allocation on computers, actuator control, nor replaces the underlying real time operating system. Its central role is shown on Figure 1.

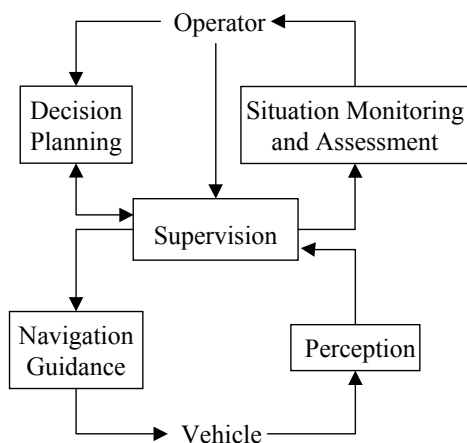


Figure 1 – Central role of the supervision function

The embedded decisional software architecture has to integrate all the data relative to the mission (its objectives), vehicle behaviour monitoring, connection to deliberative software programs, communication with the ground station and other vehicles, and reaction to disruptive events. The main features required for such an architecture are: robustness, reliability, modularity, flexibility, genericity - regarding to mission, vehicle and environment -, independence from software components, easy interfacing.

Several types of architectures exist in the literature. In this paper, we focus on the ProCoSA software package, used by ONERA for controlling and monitoring highly autonomous systems.

2. ProCoSA software package

ProCoSA, which stands for “Programmation et Contrôle de Systèmes à forte Autonomie”, was first developed in 1993 at ONERA while performing research in the field of mobile robotics. Several enhancements and its rewriting in an interpreted language led in 1999 to its official registration.

ProCoSA was designed in order to provide the developer with an integrated package putting together and synchronising the various functions achieving system autonomy, among which:

- data processing (sensor data, situation assessment data, operator’s input);
- nominal mission monitoring and control (vehicle and payload control actions);
- decision (management of disruptive events, replanning).

These functions are often developed as separate subsystems. They have to co-operate in order to fulfil the autonomous system behaviour requirements for the specified missions. More precisely, the needs are the following:

- offline tasks: specification of the nominal and non-nominal procedures, including co-operation between procedures and software programs, software program coding for embedded operation; a software program includes a set of functions that can be called by procedures;

- online tasks: procedure execution, event monitoring, and management of the dialog with the operator.

ProCoSA is based on the Petri net graphical and mathematical modelling tool for discrete event systems. It includes the following components:

- *EdiPet*, a graphical user interface for Petri nets, is used both by the developer for procedure design and by the operator for execution monitoring;
- *JdP* is the Petri net player of ProCoSA:
 - it executes the procedures: looking for the occurrence of events, it fires the event-triggered transitions of the Petri nets and runs associated actions;
 - it supervises the dialog between procedures and software programs;
 - it manages the communication with systems outside the vehicle.
- *Tiny*, a Lisp interpreter specially dedicated to distributed embedded applications, is the development language of JdP. The communication protocol for the exchange of data is socket-based (TCP/IP).

Tiny and JdP were respectively developed at the computer science and automatic control departments of ONERA. Prolexia Company developed EdiPet.

The following subsections describe:

- the Petri net formalism used in ProCoSA;
- the Petri net player (JdP);
- EdiPet;
- the property verification process.

2.1. ProCoSA Petri nets

A Petri net [2] $\langle P, T, F, B \rangle$ is a bipartite graph with two types of nodes: P is a finite set of places and T is a finite set of transitions. Arcs are directed and represent the forward incidence function $F: P \times T \rightarrow N$ (from a place to a transition) and the backward incidence function $B: P \times T \rightarrow N$ (from a transition to a place) respectively. The marking of a Petri net is defined as a function from $P \rightarrow N$, and symbolised by tokens: a given marking is associated to a given state of the system modelled by the Petri net. The evolution of tokens within the net is achieved through transition firing (Figure 2), which obeys transition firing rules:

- a transition is enabled if its input places contain at least the number of tokens given by the F forward incidence function;
- an enabled transition can be fired, and if it is fired, this number of tokens is removed from its input places;
- a number of tokens given by the B backward incidence function is added in its output places.

Petri nets allow sequencing, parallelism and synchronisation to be easily represented.

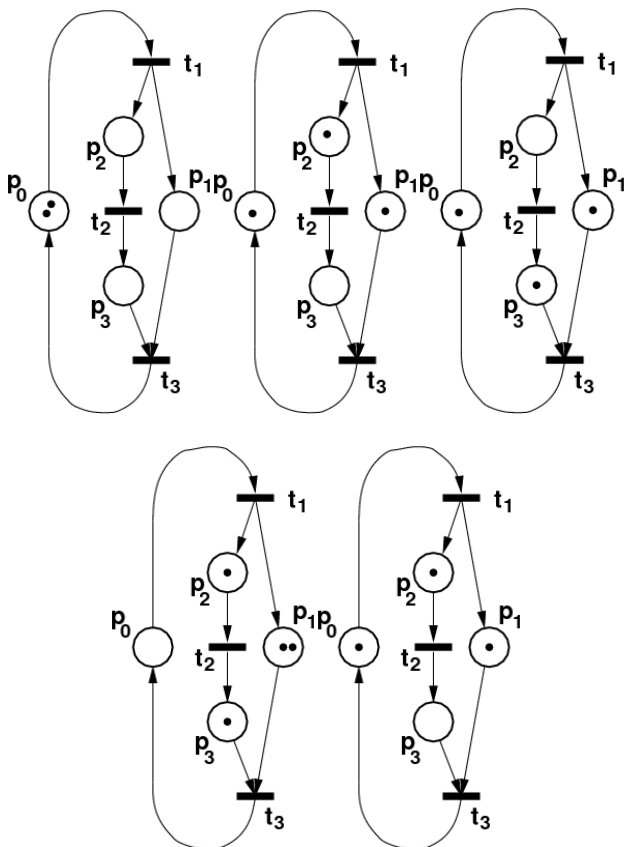


Figure 2 – Example of a Petri net transition firing sequence:
 $t_1 t_2 t_1 t_3$

Petri nets used by ProCoSA are interpreted nets: triggering events and actions are attached to transitions: an enabled transition is now fired *iff* the associated triggering event occurs, and the associated actions are executed. They also are “safe” nets: only unary arcs are used, and places should not contain more than one token. Special places, called “global places”, have been introduced in order to ease synchronisation between nets while preserving modularity: a global place is “shared” between different nets, thus enabling the behaviour modelled by a given net to take into account a state of the system modelled in another net. This feature is particularly suitable for handling disruptive events. Timers can be programmed within ProCoSA: a special action enables a timer variable to be instantiated, which allows actions with a limited duration to be modelled. Finally, the hierarchical modelling features offered by ProCoSA enable the developer to structure the whole application in a generic way: at the highest description level, nets model generic behaviours, regardless of the characteristics of a given vehicle; at the lowest level, they model the sequences of elementary actions to be performed by vehicle or payload. This modular approach eases a quick adaptation to system changes (e.g. taking into account a new payload).

Several types of actions can be associated to transitions:

- activation of a Petri net (a Petri net is activated when it receives its initial marking);
- deactivation of a Petri net (when a Petri net is deactivated, it loses its marking);
- emission of an event;

- emission of a request towards JdP (e.g. a timer initialisation);
- emission of a message towards a software program.

Several parameters can be associated to an event and used by the actions associated to the transition. This enables to establish a limited data flow between the different software programs activated by the Petri nets: when a software program ends, it sends an event towards a Petri net (usually the one that launched it) with a set of output parameters. Those parameters can be immediately transferred by the receiving transition to the next software program activated by this transition.

2.2. The JdP Petri net player

JdP was developed in Tiny language. Tiny is a Lisp interpreter designed for distributed embedded applications and includes a library implementing the TCP/IP communication protocol. An important feature of ProCoSA lies in the fact that there is no code translation step between the Petri net procedures and their execution: they are directly interpreted by the Petri net player, thus avoiding any supplementary error causes.

When a ProCoSA application is launched, JdP first reads the Petri net structures and establishes the socket connections with EdiPet (if used during the execution phase) and software programs. Specified Petri nets are activated (they receive their initial markings), and the internal JdP loop is ready to receive the incoming events.

2.3. EdiPet graphical user interface

Prolexia Company developed the EdiPet graphical user interface (Figure 3). This tool is used both for the development of a ProCoSA project and for execution monitoring. The set of Petri nets, the set of software function names and their relations define a project in EdiPet. EdiPet thus allows:

- the connections inside the whole project between JdP, nets and software programs;
- the graphical creation of Petri nets; several editor windows display and allow to modify attributes associated to each object (net, place, transition, event, action);
- the generation of relevant interfaces between Petri nets and software programs; EdiPet generates the function prototypes, which have then to be filled by the software developer;
- the display of the net states during execution; when activated (which means that one token is present), places and transitions are displayed in red.

During the execution phase, EdiPet can be used in the ground station of the autonomous vehicle, as far as a communication link is established.

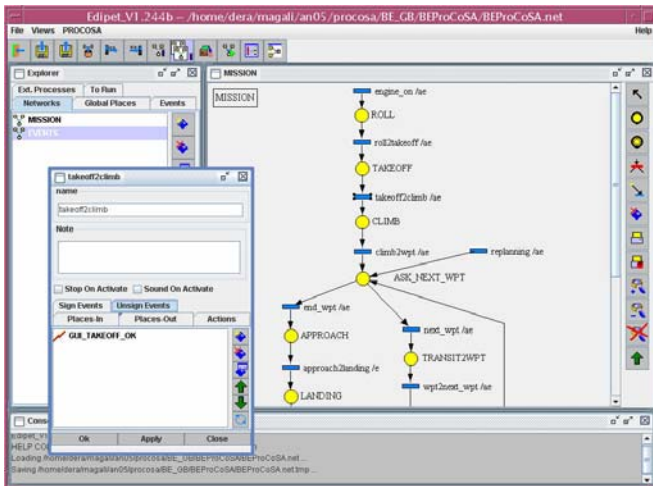
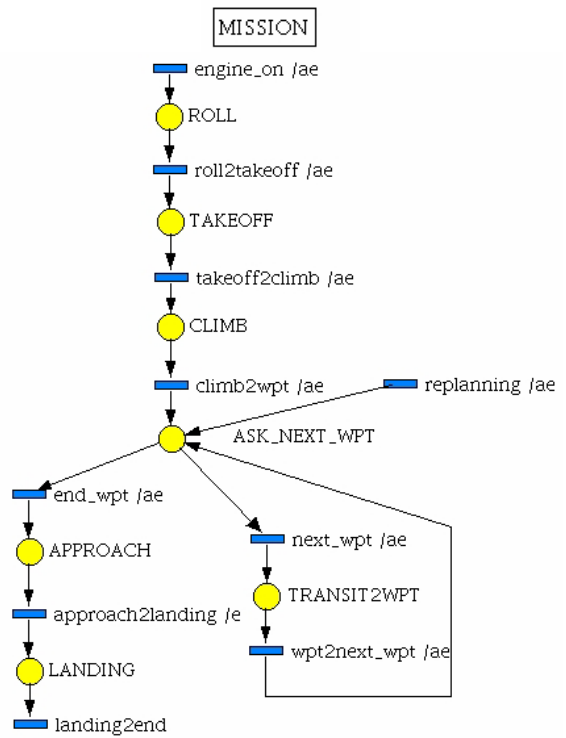


Figure 3 – EdiPet graphical user interface

The example shown on Figure 4 and Figure 5 models a simple project for the supervision of a UAV. The objective of the mission is to join a sequence of waypoints. Several payloads are available onboard, and the activation of a given payload is associated to each waypoint. The MISSION Petri net models the main mission phases: roll, takeoff, climb, transits to each waypoint, approach and landing. The GUI software program simulates the guidance of the vehicle. In the nominal execution, the DEC decisional software program gives the next waypoint to join. The EVENTS Petri net models two examples of non-nominal reactions. If a payload fails, MISSION is deactivated, DEC is called and computes another list of waypoints that do not use the faulty payload. The replanning transition of the MISSION net is then fired and the vehicle continues the mission with the new list of waypoints. In case of engine failure, MISSION is also deactivated, DEC is called and computes the nearest emergency site. The EVENTS net supervises directly the transit to this site by calling GUI.



EVENTS

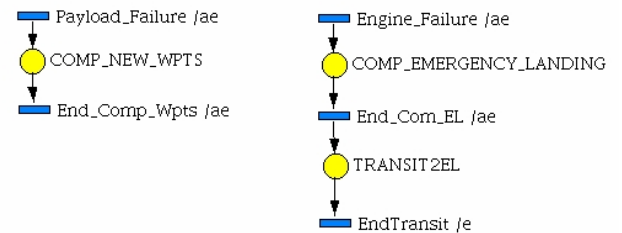


Figure 5 – Examples of EdiPet Petri nets

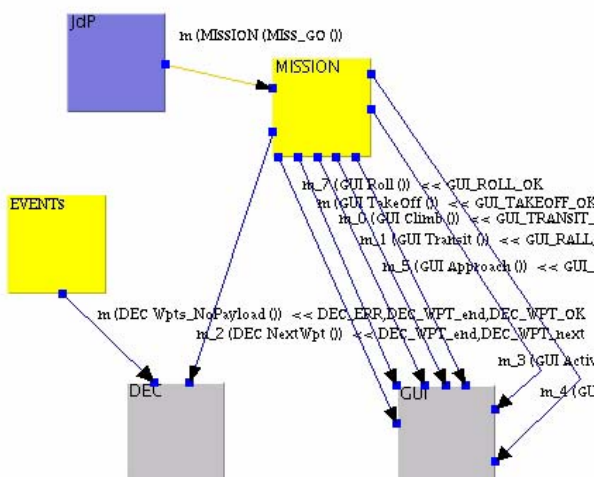


Figure 4 – Example of EdiPet project

2.4. Verification process

ProCoSA includes a verification tool, which makes use of well known Petri net analysis techniques to check that some “good” properties are satisfied by the procedures, both at the single procedure level and at the whole project level (that is to say taking into account inter-net connections).

The following properties are checked:

- place safety (not more than one token per Petri net place);
- detection of dead markings (deadlocks);
- detection of cyclic firing sequences (loops).

The principle of this analysis lies on the automatic generation of the reachability graph, which contains all the possible reachable states of each net and of the whole set of interconnected nets. As nets are safe, this set is necessarily finite, and its analysis permits to deduce the above properties.

3. Applications

Several projects are ongoing with ProCoSA:

- with DGA/GESMA on an Autonomous Uninhabited Vehicle [3];
- with EADS DS SA on an Uninhabited Aerial Vehicle [4];
- at SUPAERO, a French Engineer School, on Uninhabited Ground Vehicles, for team operation [1].

3.1. AUV project

GESMA, in co-operation with ONERA and PROLEXIA, develops a software architecture with four levels of autonomy from tele-operated mission to fully autonomous goal driven mission. Tele-operation is seen as level 0: the operator uses a control box to move the vehicle. Main tests of the vehicle were performed within this level: battery, communication, sonar, and other sensors. At the 1st autonomy level, an ordered set of elementary controls prepared by the operator describes the mission. Sixteen controls combining monitoring and modification of main variables (duration, speed, heading, immersion, and altitude) have thus been implemented. In May 2005, sea trials conducted with the Redermor AUV successfully validated the pilot software program. At the 2nd autonomy level, a mission is defined by a set of segments (straight line trajectories).

A ProCoSA based architecture has been implemented for the 3rd autonomy level: the mission is defined by a set of mission areas where a survey procedure is performed. At the end of these operations, the vehicle joins the end area. The environment is defined by bathymetry, currents, forbidden areas and non-navigable water data. The planning software program has then to compute the 2D itinerary (the order to join the mission areas), the 4D trajectory between mission areas, and the survey planning.

3.1.1. Experimental configuration and decisional architecture overview

Experiments are conducted with the Redermor AUV (Figure 6). Three computers and thirteen distributed Can interfaces with computation capabilities are installed on the platform. Serial link, Can Bus, I2C and Ethernet connections are available for payload integration and data exchange. OA1 computer is in charge of complex vehicle functions, supervision and mission planning; it thus includes the decisional software architecture. OA2 and OA3 computers are used for sonar payload controls and treatments like Computer Aided Detection and Classification algorithms for mine warfare. The embedded architecture is shown on Figure 7.



Figure 6 – Redermor AUV

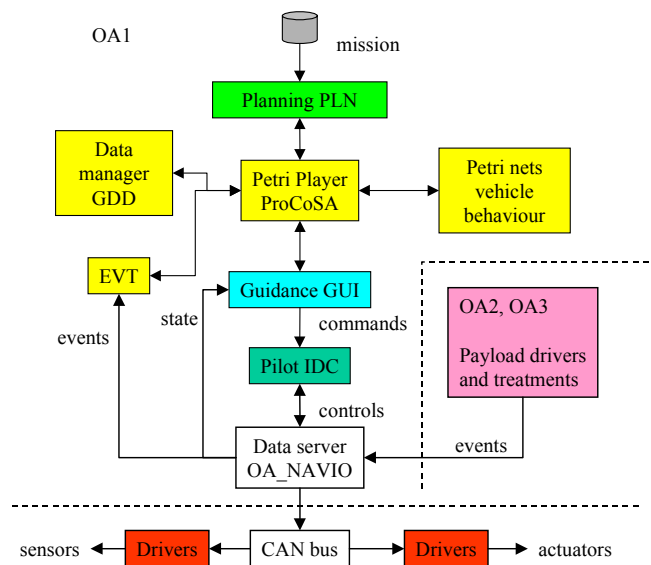


Figure 7 – AUV embedded architecture

For mission supervision, the decisional architecture in OA1 computer includes:

- the Petri net player of ProCoSA;
- vehicle behaviour modelling through Petri nets, for nominal and non-nominal situations;
- four software programs connected to ProCoSA: planning (PLN), guidance (GUI), dynamic data manager (GDD) and event listener (EVT);
- the pilot program software (IDC) that computes controls sent to the engine;
- the data server (OA_NAVIO) developed by GESMA that carries out bi-directional communication links with the hardware architecture.

3.1.2. Nominal scenario

The behaviour of the vehicle during the execution of a mission is described in eleven Petri nets. This description is hierarchical (Figure 8):

- In Mission net, at level 1, two places model the stop and the ongoing states of the mission;
- Mission_Phases net at level 2 models main phases of the ongoing mission: planning initialisation, the loop structure enabling the vehicle to join each mission area (transit to the area and survey) and transit to the end area;

- Three Petri nets model level 3: Transit_to_Area net for transiting to the next mission area and Operation net for survey achievement; Initialisation net runs itinerary and operation planning when starting the mission;
- Level 4 is devoted to computation: Itinerary_Planning net computes an itinerary for the saved mission taking into account non-navigable areas; Trajectory_Planning net computes a trajectory between two areas modelled by their centroid waypoint; Operation_Planning net asks for vehicle state and computes the operative sequence; both a trajectory and an operative sequence are composed of linear trajectory followings and course changes;
- Level 5 executes the mission: Trajectory net asks for the next trajectory and runs it; Survey net asks for the operative sequence and runs it; Planning_and_course_change net computes the required gyrations and heading following sequence and executes it.

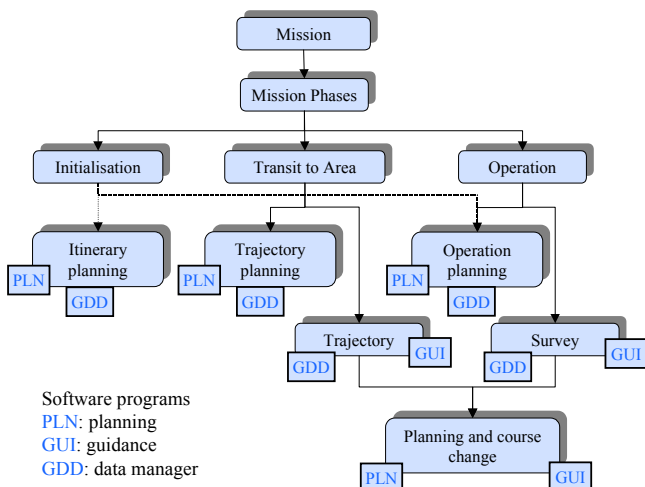


Figure 8 – Hierarchy of Petri nets

3.1.3. Non-nominal scenario

Many events can affect AUV missions and require onboard replanning. At present, three types of events are implemented:

- an alarm event forces the vehicle to move directly to the end area: a new itinerary to avoid non-navigable areas and a new trajectory are computed;
- when arriving on an objective area, the real current is different from the predicted one and invalidates the already-computed survey: a new operative sequence is thus computed;
- the operator asks for a local operation of inspection, for example to inspect a suspicious object. A specific operative sequence is planned before the vehicle resumes its mission.

These events are considered in the architecture through three new Petri nets. The Decision net implements the decisions that the vehicle must make according to the type of event, e.g. return to the end area in case of an alarm event. The Action net executes the decisions; it can run nominal nets. The Inspection net executes the inspection asked by the operator.

3.1.4. Lab bench tests

A bench test has been developed to test the whole decisional architecture. The OA_NAVIO data server has been connected to on the one hand a Redermor simulator, and on the other hand the IOVAS interface. The simulation of several missions allowed to validate the desired behaviour of the vehicle (Petri nets), the decisional functions of PLN, the management of dynamic data in GDD, the guidance and the pilot of the vehicle (GDD and IDC) and the reception of disrupted events (EVT) together with the supervision on IOVAS operator interface (Figure 9). Nominal and non-nominal scenarios were both successfully simulated.

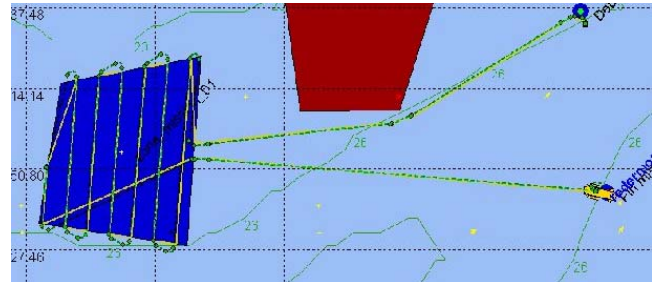


Figure 9 – Supervision of a simulated AUV mission. Survey area is blue, forbidden area is red, planning trace is yellow, vehicle simulated trace is green.

3.1.5. Sea experiments

Recent sea experiments have been conducted in Douarnenez Bay. Three missions were carried out. The vehicle is followed by acoustic means, and only a few points are currently available (Figure 10). Vehicle immersion, transit to the survey area, line following at a given altitude and return to the end area were successfully performed. These experiments validated the embedded use of ProCoSA. Emphasis should now be put on guidance accuracy, perception function, classification of disruptive events, situation monitoring and assessment functions.

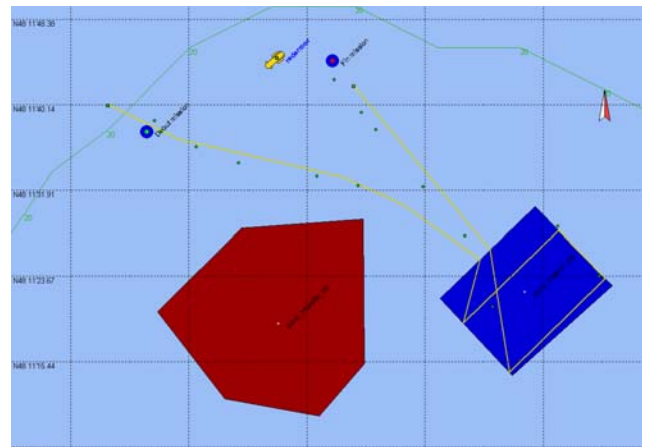


Figure 10 – Supervision of a real AUV mission. Acoustic vehicle trace is green.

3.2. UAV project

EADS and ONERA are involved in a national project that aims at testing an architecture designed for mission supervision in a UAV and demonstrating the relevance of such architectures in future uninhabited vehicles. As all categories of UAVs have to perform their missions in complex environments with the same types of constraints, the embedded architecture has to be generic, i.e. not dedicated to a given mission, environment or vehicle. As the mission may be disrupted by internal or external events, e.g. failures, weather situation, interfering aircraft, and threats, onboard plan monitoring and replanning are required in order to deal with nominal or disruptive events, avoid systematic return to base and proceed with the mission as well as possible given the new constraints.

3.2.1. Experimental configuration and decisional architecture overview

Experiments are conducted on a light plane, a Dyn'Aero MCR-4S (Figure 11). Two computers are devoted to the control part of the plane, and a third one to the decision part, i.e. mission management. The first control computer is directly linked to the plane sensors and actuators (e.g. the automatic pilot) and to the ground station, while the second one acts as an interface between the previous real time control computer and the decision computer: it sends formatted frames and interprets elaborated orders.



Figure 11 – Light plane used for experiments in UAV project

The role of the software decisional architecture implemented on the decision computer through ProCoSA (Figure 12) is thus to monitor the main mission phases of the nominal scenario, to manage the dialog with the operator (payload use), and to generate control decision when disruptive events occur. In order to elaborate the pre-specified events used by ProCoSA from the telemetry frame data, an additional interface software layer was developed (Figure 13).

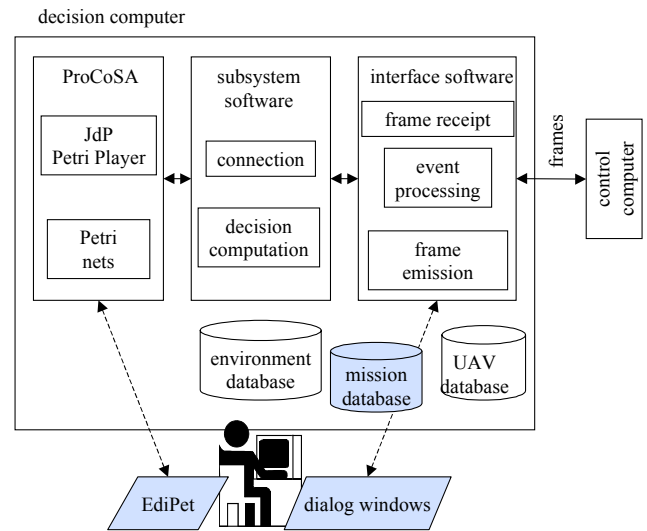


Figure 12 – UAV embedded decisional software architecture

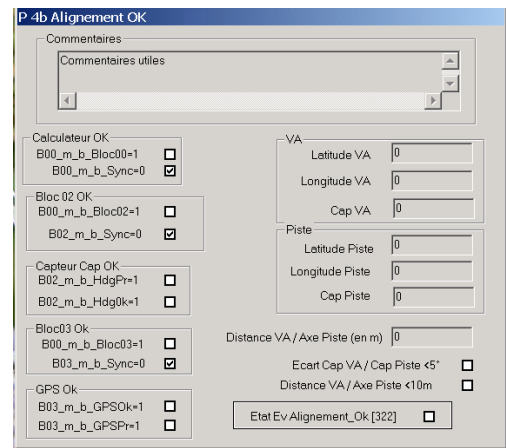


Figure 13 – Example of the “ready for takeoff” event elaboration

3.2.2. Nominal scenarios

A four-level mission modelling architecture was defined in order to guarantee a generic approach:

- level 0: initialisation of the communication protocols between ProCoSA and the other software layers;
- level 1: global state of the mission and modes monitoring (nominal - non nominal);
- level 2: main nominal phases of the mission (from pre-flight tests and takeoff to touchdown and end-of-flight tests);
- level 3: sub-phases of the mission.

Level 3 corresponds to less generic procedures, i.e. more specific to the vehicle type or to mission and payload characteristics. The Petri net shown on Figure 14 details the linking of the different steps within the operational area: this net clearly shows the looped structure enabling the set of pre-programmed tasks to be achieved, and includes communication requests to the operator. ProCoSA timers are used to limit the time allowed for the operator’s answer.

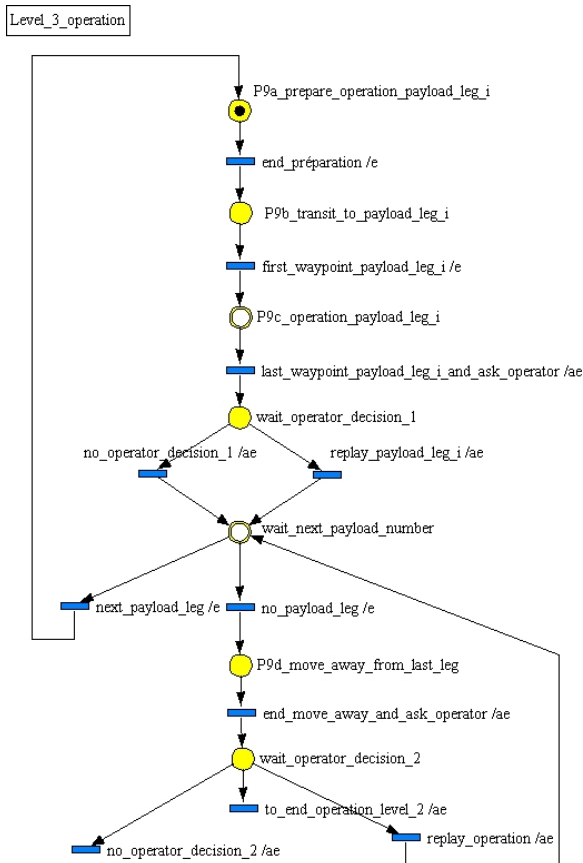


Figure 14 – Modelling of the linking of operational tasks

3.2.3. Non-nominal scenarios

In order to be able to apply a generic approach to deal with disruptive events, they were classified in four categories:

- *catastrophic* events lead to mission abortion, and cannot be recovered; when such an event occurs, the processing of any other kind of events is aborted and no further incoming event can be processed; example: engine total failure;
- *safety-related* events lead to modifying the flight profile or the flight plan - e.g. change route for a while - which may induce delays or new constraints on the use of the payload; examples: interfering aircraft, new forbidden area, turbulence...
- *mission-related* events only have consequences on the mission itself; replanning amounts to adapt the mission to the new constraints, e.g. remove waypoints; examples: camera failure, violated temporal constraint, new mission goal...
- *communication-related* events are related to communication breakdowns between the UAV and the ground; such events result in the UAV being fully “autonomous” therefore it has to proceed with the mission as planned; example: telemetry failure.

According to this classification, one Petri net was designed for each disruptive event category: an example is given by the engine failure Petri net shown on Figure 15: one can note the use of the ProCoSA “global places” feature (see section 2.1), which enables to adapt the

reconfiguration actions to the current state of the mission. This reconfiguration process is achieved through software function activation requests, which enable to build a set of control orders to be sent to the control computer

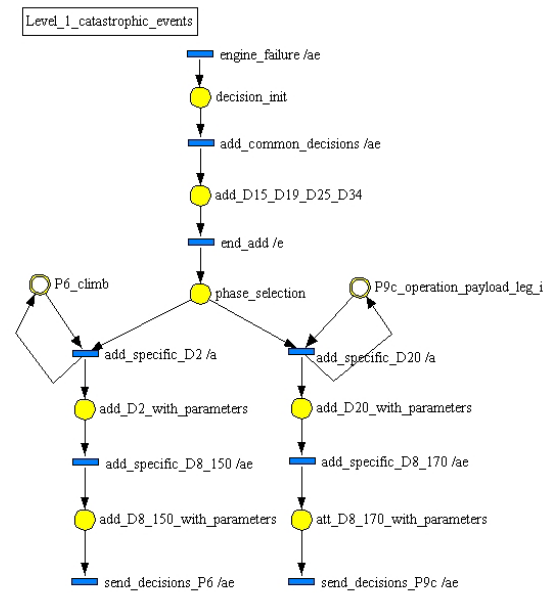


Figure 15 – Engine failure reaction modelling

3.2.4. Ground and flight tests

Two series of field tests are planned in March and May 2006. A test series will be organised as a two-step process: during the first week, ground tests will be conducted in order to prepare and simulate the scenarios, which will be run on the plane. Flight tests will be conducted during the second week, with a pilot and an operator onboard the plane.

The nominal scenario will be tested first. Non-nominal scenarios implying a unique disruptive event will be considered afterwards, and eventually scenarios including two or three cumulative disruptive events.

A double check process will be achieved during each flight test. Flight data (telemetry frames) and the corresponding Petri net states will be registered onboard. The ProCoSA layer of the decision computer will be duplicated on the ground station that will also receive the real-time telemetry frames, thus enabling system state to be monitored on the ground.

3.3. UGVs project

The computer science and automation lab (LIA) at SUPAERO and the Systems Control and Flight Dynamics Department (DCSD) at ONERA are involved in a co-operation on mobile robotics to answer a national need to integrate robots into military operations. The project goal is to operate several autonomous robots. In these studies, the choice of a centralised architecture was made.

3.3.1. Experimental configuration and decisional architecture overview

Robots used in the project are Peeke robots, developed by Wany Robotics (Montpellier, France). They feature

three individual racks for computer cards (for communication *via* WiFi, and/or camera and image management), a shock detection sensor and are surrounded by an Infra Red sensor ring (Figure 16).

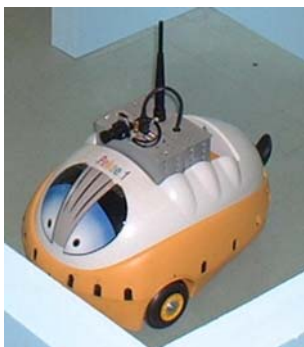


Figure 16 - A Pekee robot. Note the WiFi antenna, the IR sensor ring and the camera, as well as two occupied racks

Two libraries were developed:

- the movement library stores elements such as free translation and rotation but also half-controlled displacements such as translation until obstacle or controlled movements such as following a wall, a corridor, a sinuous route...
- the picture library is based on openCV primitives and allows in particular the detection of obstacles and localisation of markers.

These library elements constitute a set of services that the agent may use in order to achieve the mission goals.

Several groups of students worked on these experiments. The last objective was to implement the control of the basic moves for two robots in a known environment. The mission (Figure 17) consists in virtually changing the position of several coloured rings following a defined order. As robot moves are independent, a planning algorithm was developed to manage area occupancy conflicts.

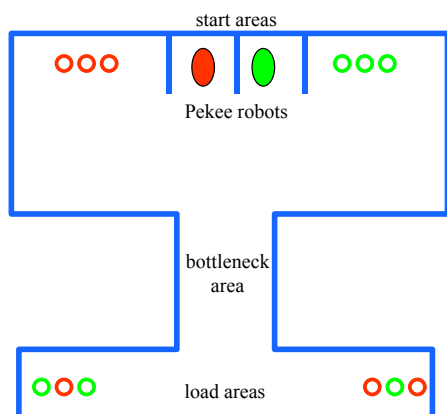


Figure 17 – UGVs mission

The control architecture for each robot heavily relies on ProCoSA. The architecture is composed of three layers (Figure 18):

- the ProCoSA layer models actions chronology; it is centralised;

- the interface layer translates ProCoSA orders into robot controls and robot sensor data into ProCoSA events; it is written using the Visual C++ development tool; controls are mainly related to speed and heading;
- the robot layer executes controls and sends sensor data coming from IR sensors and camera image.

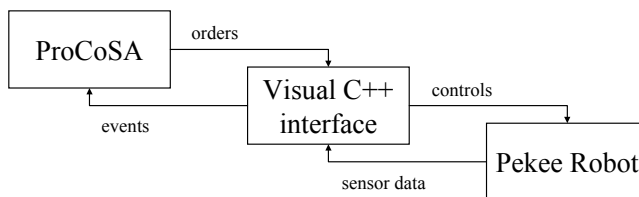


Figure 18 – UGV architecture

3.3.2. Lab tests

Eight elementary moves have been modelled: they allow the robot to move in the labyrinth and to take into account the bottleneck area (go forward, go backward, turn right, turn left, follow right wall, follow left wall, enter bottleneck, exit bottleneck). Only one robot could enter this area at the same time. Seven Petri nets were developed, three per robot and one for their synchronisation, i.e. the management of the conflict in the bottleneck area according to the plan algorithm.

The mission was successfully executed by robots. This validated the use of ProCoSA for synchronising a two-robot mission.

3.3.3. Generic supervision approach

Some work [5] is carried out on designing a more generic supervision architecture that would take advantage of the modular nature of the robot and run the controller and the diagnosis module (Figure 19).

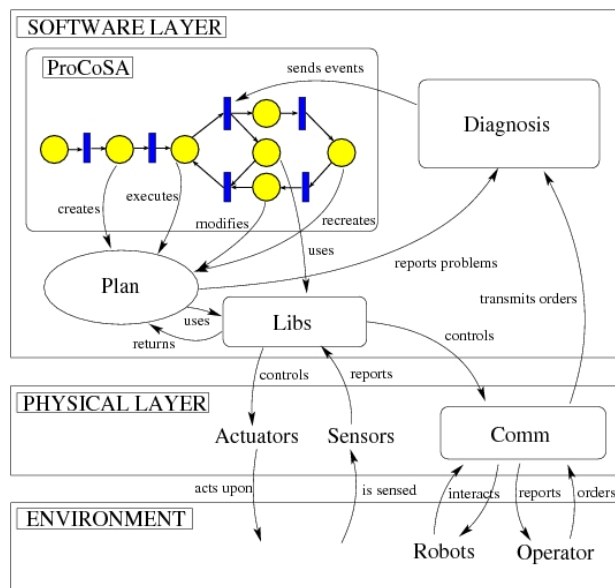


Figure 19 - Robot embedded architecture

Figure 20SEQARABIC proposes a Petri net model for the controller. The initial planning creates the plan. The execution phase runs the plan that in its turn executes actions from the libraries. A replanning step is triggered at

the occurrence of a disruptive event: the robot is set in a reaction mode (safety mode) while the event is analysed. The consequences of the event are then dealt with during the repair that recreates the plan. Once the repair is calculated, it is adjusted so as to smoothly switch from reaction to repaired plan execution.

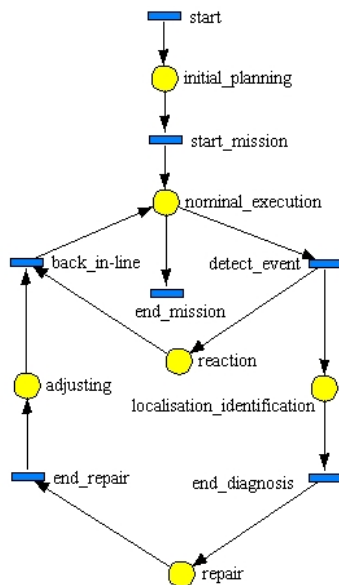


Figure 20 - ProCoSA plan controller

3.3.4. Future experiments

The mission will consist in a search and rescue operation in a partially known urban environment. The team is composed of two aerial robots and two terrestrial robots. The robots have knowledge of possible routes to move in the environment. The uncertainty parameters, such as obstacles barring expected paths, robot or service failure... are handled through event firings that trigger replanning phases on parts of the plan.

The experimental set-up uses four Pekee robots, a labyrinth modelling the environment and a transparent pane to bear the “aerial” robots (Figure 21). The two aerial robots are characterised by a downward camera in order to detect ground obstacles and deliver accurate global localisation for ground robots. All communications are WiFi-based.

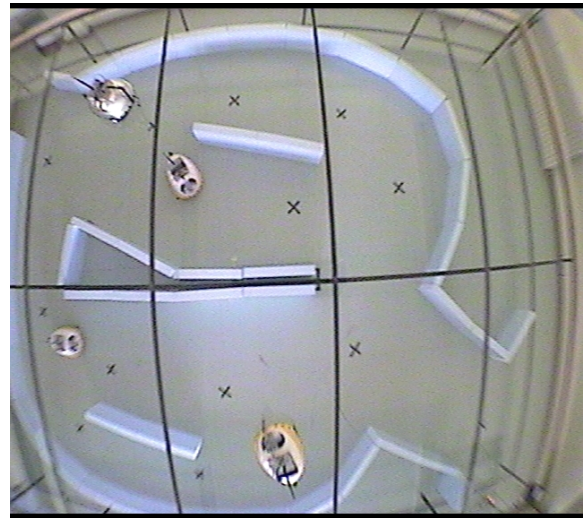


Figure 21 - Experimental set-up. Note the two aerial robots on the Plexiglas pane with downward camera

4. Conclusions

The current version of ProCoSA allows designing decisional embedded software architecture: Petri nets describe the execution logic for the various specified autonomous vehicle behaviours, including both nominal mission phases and reactions to disruptive events. It also manages connections with software programs associated to specific functionalities such as situation assessment, planning, guidance. Those behaviours are directly interpreted and executed by the Petri net player, without any intermediate code translation step, and on-line execution monitoring is possible with EdiPet. Significant validation steps can be achieved during the design phase thanks to Petri net formal analysis properties.

ProCoSA software has been successfully used in several projects for the control of autonomous vehicles. First sea experiments validated its use in embedded architecture. Aerial tests are planned by the end of this year. Research on its implementation in several mobile robots composing a team is ongoing.

The main objective of the proposed embedded decisional software architecture is to supervise mission execution whatever occurs. It thus deals with environmental uncertainties: reactions to disruptive events are implemented in Petri nets that call deliberative tasks. Deliberative tasks and mission data are independent of ProCoSA and that point offers modularity and genericity to the whole architecture. Indeed, specificity of the vehicle is taken into account in databases and in the interfaces connected to the control computer.

Current results point out several possible ways of improvement:

- a perception function that studies and develops methods to elaborate and update real world sensor data, would help to take appropriate decisions; of course, good data quality is required as well;
- a situation monitoring and assessment function that estimates the system parameters and predicts their

evolution could help to anticipate the arrival of disruptive events; this should also increase security level for the vehicle; image processing is also a difficult task to implement onboard;

- a generic study of all types of events, their classification and identification of associated reactions are necessary in all autonomous system, as emphasised in UAV experiments;
- all studies drew attention on the necessity of enhanced planning algorithms for autonomous vehicles; research have to be conducted to improve proposed algorithm with regard to duration constraints; mission objectives could also be selected onboard (objective planning function) according to collected environmental data;
- all possible communications between the ground operator and the vehicle have to be defined properly, especially the operator's decisions and the associated reactions onboard. This communication protocol gives the vehicle its level of autonomy. An onboard architecture adapting its autonomy level according to the types of disruptive events could also be considered;
- simulation remains essential to valid autonomy architecture before its implementation; the use of a bench test before sea experiments in the AUV project led to architecture validation during the first autonomous missions;
- studies on the collaboration between several autonomous vehicles have to continue, as this is the main point in future operational theatres;
- the operator's role evolves as the autonomy level increases, and ground systems have to evolve as well, e.g. with implementation of decision support systems.

References

- [1] B.T. Clough, *Metrics, Schmetrics ! How the heck do you determine a UAV's autonomy anyway?* Performance Metrics for Intelligent Systems Workshop, 2002, Gaithersburg, MA, USA.
- [2] Murata, T. (1989). *Petri nets: properties, analysis and applications*. IEEE. 77 (4), pp. 541-580.
- [3] F. Dabe, H. Ayreault, M. Barbier, S. Nicolas, *Goal Driven Planning and Adaptivity for AUVs*, UUST 05 Unmanned Untethered Submersible Technology, 21-24 August 2005, Durham, NH, USA.
- [4] M. Barbier, J.F. Gabard, J.H. Llareus, C. Tessier, J. Caron, H. Fortrye, L. Gadeau, G. Peiller, *Implementation and Flight Testing of an Onboard Architecture for Mission Supervision*, 21st IUAUS International Unmanned Air Vehicle Systems Conference, April 3-5, 2006, Bristol, UK.
- [5] O. Bonnet-Torrès and C. Tessier, *Cooperative Team Plan: Planning, Execution and Replanning*, AAAI'06 Spring Symposium on Distributed Schedule and Plan Management, March 2006, Stanford, CA, USA.