Connectivity of Thetis, a Distributed Hybrid Simulator, with a Mixed Control Architecture

Olivier Parodi, Abdellah El Jalaoui, David Andreu

Connectivity of *Thetis*, a Distributed Hybrid Simulator, with a Mixed Control Architecture

Olivier Parodi, Abdellah El Jalaoui, David Andreu
LIRMM - University of Montpellier 2 - CNRS
161 rue Ada 34392 Montpellier, France
Email: {parodi, eljalaoui, andreu}@lirmm.fr

***Abstract*—The purpose of this paper is to present the linkage of *Thetis* (a real time multi-vehicles hybrid simulator for heterogeneous vehicles) with a control architecture for the management of contextual tasks on AUVs (*Autonomous Underwater Vehicle*), both developed at LIRMM (France).**
An AUV is mainly composed of an onboard informatics system and a set of sensors and actuators. We design a 3-layer control architecture (global supervisor, local supervisors, and a set of management modules which are in charge of the means of perception and actions), allowing for facing Hardware (embedded instrumentation) and software evolutions. Due to the complexity and the sophistication of the control of actual underwater systems, it is necessary to check the embedded robot controller, i.e. all its constitutive hardware and software sub-systems before beginning any real mission. To achieve this goal its possible to call upon simulation, thus avoiding heavy and numerous experimentations.
***Thetis* is able to provide the necessary environment to test real robots and their interactions (communication and collision detection). Indeed this simulator allows hardware in loop simulations with the support of virtual sensors.**
The paper first describes the architecture and functionalities of *Thetis*, then the control architecture of our AUVs is briefly presented. We conclude with the advantages of connecting the 2 systems together.

I. INTRODUCTION

The need to operate in ever deeper waters and to reduce the costs of missions, has brought researchers to concentrate on the elaboration of autonomous vehicles which are able to fulfil tasks which have until recently needed a human operator. The need of autonomy in a environment which is in perpetual evolution requests from the vehicle a capacity to estimate its state and the environment state. The computing power, the miniaturization and the fall of consumption of the computers allow to imagine architectures increasingly sophisticated to assume scenarii increasingly more complex. In parallel the number and the complexity of tests necessary to validate this kind of architecture is growing up. Thus simulation tools play an important role: they are able to help us in these tests in order to validate a command or a software architecture.
These technologies limit the human ressources, the difficulty of real experiments, the cost and the time spent. There are different sorts of simulators. A useful classification is proposed in [1]. Simulators can be classified in 4 categories: the offline simulators, the online simulators, the hardware in loop simulators and the hybrid simulators.
Offline Simulators permit to start designing the control of robots in a first approximation. Matlab/Simulink is a good tool for this kind of simulation. Indeed there are a lot of available toolboxes for robots and in particular for AUVs. In [2] such a toolbox is presented; it allows to implement the mathematical models quickly. But we have to keep in mind that the temporal aspect of the simulation is not taken into account and it potentially makes the algorithm inoperative when it is transferred on a real robot and thus the command is not really validated.
Online Simulators belong to another type of simulator which allows to take the temporal consistency of the simulation into account. Indeed, in this type of simulation one second of simulated time actually corresponds to one second in real time. However the algorithms are still not executed on the robot itself and the temporal behavior of the computer used for the simulation can appear different from that which will be used to control the robot.
In *Hardware In Loop (HIL) simulators* the control is executed on the robot itself but the commands given to the actuators are directed towards the simulator instead of the real robot or at the same time as on the real robot [3]. The commands to the actuators are directed to the simulator which is therefore in charge of the model evolution, instead of or at the same time as on the robot. However, in this sort of simulation the external world is not taken into account. Only the proprioceptive sensors are used and all the algorithms and onboard systems can't be fully tested.
Hybrid simulator are HIL simulators where real and virtual systems interact together in an augmented reality. It is therefore necessary to simulate an environment (static or dynamic) in which the robot will be able to operate all its systems. It's possible to test all the algorithms of the machine from low level control of sensors or actuators to the whole architecture. This approach has been used by several authors such as [1] in which, authors present Neptune, their real time graphic multi-vehicles simulator, permitting to perform online, HIL and hybrid simulation.
So first we'll present the architecture of our simulator: *Thetis*, specifying the models we use. Then we'll expose the control architecture used on our AUV: Taipan. Finally we'll show how and why it is useful to connect the 2 systems.

II. *Thetis*: HYBRID SIMULATOR

Thetis is a real time hybrid simulator permitting to consider heterogeneous multi-vehicles scenarii. This type of simulator represents a real challenge to favor the research on vehicles cooperation.

A. Critical concepts

This simulator is composed of a set of mechanisms which allow the exploitation of simulation with enough fineness to stand comparison with the real world but without replacing it in any way. Among its concepts, one of the most important is the capacity of the simulator to ensure a *temporal decoupling* between the control loop and the simulation loop. This ensures that if the controller doesn't react at the appropriate time, we'll observe a divergence on the robot's behavior. Indeed, there is no logical synchronization between the simulation loop and that of the controllers of the robots.
Another important concept is the *upgradability* of the simulator. We are able to add some new components (sensors, vehicles...) in a very easy way. This warrants us the possibility of using this simulator with different sorts of vehicles. Indeed, the *modularity* of this simulator favors the interventions of different actors. Thus, a sonar specialist will be able to implement a sonar model, to transfer it onto another computer if he considers it is too resource-consuming, all that without having to feel concerned about the simulator global functioning and without having to deeply intervene in the simulator code.
Finally, a very interesting aspect of this architecture is its *portability*. Indeed it is able to work with different robots and thus connected with different control software architectures. We only need to adapt the interface between the 2 systems (simulator and controller), in order to make them work together ensuring the frames encoding/decoding. Another advantage of the portability is the simulator's faculty to be divided and executed on different computers. This avoids to overload computers and to affect the real time capability of the systems.

B. Implementation and technical considerations

All the concepts proposed in the precedent section are supported by using 3 mechanisms and an architecture based on 3 simulators. First an *XML-based specifications exchange data* (*XML for Extensible Markup Language*) provides a high accuracy degree and structuration of the parameters of the different models used (modem, radio, fins, motor, ...) while promoting the modularity and the portability; we are not frozen on a standard. Indeed we can talk about modularity because each component of the system is described in an XML file in which all parameters of the said component are detailed; it allows to change the components of the system in a very easy way. For example, if we consider an inertial unit, the model used in an aerial drone and in an AUV will not necessarily be the same. By contrast, what will remain the same, will be the model of the inertial unit used; as for the parameters of each unit they will be described in 2 separate XML files. Now if we want to interchange the units, it's possible by calling a XML file in place of an other. All the system components work in

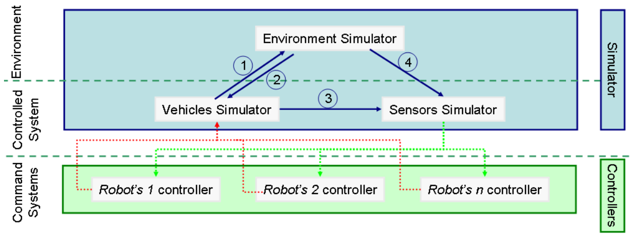

Fig. 1. Simplified architecture of *Thetis*: there is a logical sequencing between the 2 simulators even if there are independent processes. The cycle duration of this sequence must be largely lower than the cycle period of the controller.The temporal decoupling is only effected (and that's enough) between the simulator and the controllers).

the same way (fins, acoustical modem, GPS, propeller,...). Moreover, many systems could use the same formalism to describe the parameters of a component model without using all of them. For example, we can imagine that an underwater communications specialist will need a very accurate model describing its modem; he should use all the parameters of the XML file. Another user, who doesn't need such an accuracy level, will use the same XML file, but only a part of these parameters in order to animate its own model. The validation and extensible properties of the XML language make it an ideal base to enrich the model parameters files.
Thus the robots (sensors, actuators, communications etc..) used in this simulator are described in XML formalism. To do this, we suggest a set of different tags allowing to interchange and to describe the components of the robots in a easy way. Then portability and temporal decoupling is favored by using *sockets* and *local shared memories* widely. Thus it's possible to run several processes on different computers each of them interacting with others using these mechanisms. Finally we have created a set of librairies containing a set of classes enabling us to build the various objects of the simulation system. All these classes are documented with doxygen tool [13] and will be soon available online (www.lirmm.fr/∼parodi/thetis). So as to ensure performances in real time as well as effective temporal decoupling, the simulator is in fact an application distributed onto 3 simulators. The first one is a vehicle simulator which allows the simulation of the robots dynamics. The second one is a sensors simulator allowing the simulation of the various sensors of the robots. Lastly the third one is a static environment simulator (no evolution with time at yet). It allows the use of exteroceptive sensors to announce the robot/robot or robot/ground collisions. All these simulators are interconnected on a dedicated UDP/IP network. The connections between these various blocks are detailed and explained on figure 1. Only the sensors and vehicles simulators are connected to the real robot, the environment simulator being connected only to the 2 other ones. All these simulators work under linux RTAI. Information about network configuration are described in a shared XML file. All the XML files describing the components of the system are loaded at the initialisation and thus allow to instantiate the different objects

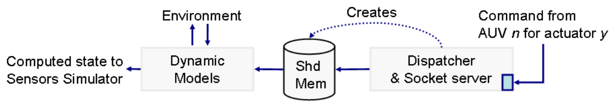

Fig. 2. Architecture of the vehicles simulator

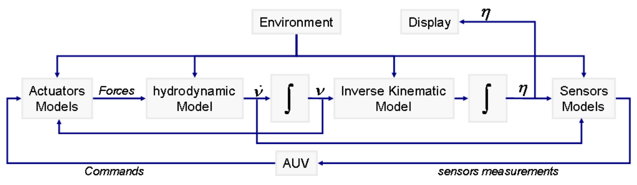

Fig. 3. Simulation cycle of the vehicles simulator

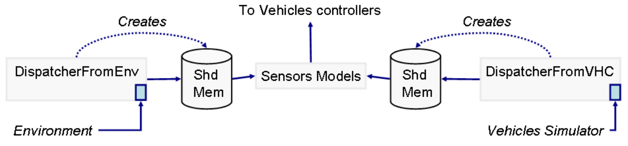

Fig. 4. Architecture of the sensors simulator

of the simulator.

C. Vehicles simulator

The vehicle simulator is in charge of the robots dynamic model evolution. We present this simulator structure on figure 2. It is composed of independent processes communicating via a local shared memory. Before starting the simulation, this simulator connects itself to the environment simulator to obtain all the physical constants (ν, ρ, g, ...). In fact, this simulator is composed of 2 main processes. The first one is the *Socket Server and Dispatcher* process, which is in charge of ensuring the communication between the simulator and the robots controllers. The robots send actuators commands calculated by the onboard computer to the simulator through UDP socket. Then the received data are decoded and extracted to a shared memory initially created by this process. The second one is the cycle of simulation. During this cycle all the forces applied on the robots are computed in order to determine the accelerations and then robots attitudes and velocities after an integration step. The computed data are sent to the environment simulator in order to verify the non collisions of the objects and if necessary to correct positions. Finally the computed data are ready to be sent to the sensors simulator. This cycle is presented on figure 3. The initial position and attitude of the vehicles are determined at the initialisation, then the dynamic model evolves according to the commands sent by the AUV and the environment.

D. Sensors simulator

The sensors simulator is in charge of providing the real robot with virtual data issued from the simulation. Presently only the models of some proprioceptive sensors are implemented. This simulator is based on the execution of 3 independent processes, communicating via 2 shared memories.The structure of this simulator is presented on figure 4. There are 2 dispatcher processes (*DispatcherFromVHC* and *Dispatcher-FromENV*) which are respectively in charge of listening to messages from vehicles simulator (velocity, accelerations, attitudes...), and messages from environment simulator (parts of maps determined according to the position and the range of the robot's sensors), and of decoding and extracting data into 2 shared memories (one for each process). The third process *SimuCAP* is the sensors simulator which elaborates answers from each sensor for each robot.These answers are computed according to each sensor model whose parameters are described in the XML file. They are elaborated according to the sensor characteristics, to the situation data issued from the vehicles simulator and lastly to the data issued from the environment simulation. Once these answers are computed, they are sent by name to each robot of the flotilla. We also call sensor, the communication devices of the robots. We are currently implementing a messaging distribution mechanism sent by the robots between them. This mechanism takes into account the delay, the flow and the losses caused by the type of communication device in use, and the propagation medium. It will permit to test some command laws for coordinated AUV flotilla.

E. Environment simulator

The environment simulator is in charge of producing geophysical maps around a given geographical point, detecting the collision between 2 robots or between a robot and the ground, and finally computing the new positions of the colliding robots. These functionalities are being implemented and are architectured around 2 independent processes also communicating via a shared memory. It does not yet integrate a real environment (and its obstacles).

III. Models and assumptions

In this chapter we explain the modeling methods which we have chosen to develop the simulator. Although it is not exclusively limited to simulate AUVs, it is the first model that we have implemented because our team works on this type of robot [4]. Other models will be implemented later if needed. Hence, the simulated sensors suite is dedicated to submarine applications and the modeled environment is exclusively underwater. Obviously, all this can be easily modified in order to deal with heterogeneous robots and environment like coordination between AUVs, drones and surface crafts.

A. AUV modeling

The robot model is used to compute the robot movements according to the command vector. The modeling of the AUV, is made up of the hydrodynamic, hydrostatic and dynamic phenomena of the robot on the one hand, and of the actuators model on the other hand. Here is a brief description of these models:

- *Hydrodynamic forces*: the simulator uses the 6 dof (*Degree of Freedom*) non linear equations of 6DOF AUV expressed in the body fixed frame [5]

$$\dot{\nu} = (M_A + M_{RB})^{-1}\big(\tau - (C_{RB} + C_A + D)\nu - g(\eta)\big)$$

where:

  - ν denotes the system velocities (linear and angular)
  - M_{RB} and M_A are the inertia matrix and the added mass matrix
  - τ denotes the force and torque produced by the thrusters and control surfaces
  - C_{RB} and C_A the rigid-body, the added Coriolis and centripetal matrixes
  - D denotes the damping matrix
  - g denotes the gravity and buoyancy force and torque
  - η denotes the position and orientation vector

External disturbances (waves, oceanic currents, wind...) are not implemented yet. Moreover the potential damping, the skin friction and the wave drift damping are not considered. Only the damping due to vortex shedding is computed. Once the estimation of the accelerations is computed, we integrate a first time to obtain the velocities in the body fixed frame. It is then necessary to integrate a second time after shifting the frame so as to obtain the robot position and attitude in the inertial frame.

$$\begin{bmatrix} \dot{p}^n \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} R_b^n(\Theta) & 0_{3\times3} \\ 0_{3\times3} & T_\Theta(\Theta) \end{bmatrix} \begin{bmatrix} v_o^b \\ \omega_{nb}^b \end{bmatrix} \tag{1}$$

- *Thruster Model*: A steady state model is used for the thruster of the AUVs. We can consider that the response is instantaneous according to the input. We use a bilinear model ie a non linear function computing the thrust according to the angular speed of the propeller ω and the linear speed of the robot in the thruster direction. See [1] for more details.

$$T = C_{T\|\omega\|\omega} \times \omega - C_{T\|\omega\|\nu} \times \nu \tag{2}$$

Other authors propose dynamic models [6], [5]. The possibility of implementing such thruster models will be studied later.

- *fins Model*: the robots Taipan [4] have a cylindrical shape and are equipped with rudder at the stern and with 2 pairs of diving planes located at the bow and at the stern. The lift F_z is the projection of the resultant F on the axis, orthogonal to the fluid direction. The drag force is the projection on the axis, parallel to the fluid direction. These forces are modeled according to the following equations, expressed in a frame fixed on the control surface ([7] and [8]):

$$\begin{aligned} F_{cz} &= \frac{1}{2}\rho S V_0 C_{zs} \\ F_{xz} &= \frac{1}{2}\rho S V_0 C_{xs} \end{aligned} \tag{3}$$

where

  - ρ is the density of the fluid
  - S is the projected area of the fin, perpendicularly to the fluid direction
  - V_0 is the relative velocity of the body with the fluid
  - C_{zs} is the lift coefficient corresponding to the axes of the surface
  - C_{zs} is the drag coefficient corresponding to the axes of the surface

Once the physical action of the planes is expressed, we consider that the axis of rotation of the planes is situated at a distance d_a from the origin of the local frame of the robot. The forces of the lift and drag,no evolution with time at yet and the induced moment are therefore given by:

$$\begin{bmatrix} F_x \\ F_z \\ M_q \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}\rho S_s V_0^2 (C_{zs}\sin\delta + C_{xs}\cos\delta) \\ -\frac{1}{2}\rho S_s V_0^2 (C_{zs}\cos\delta - C_{xs}\sin\delta) \\ F_z(lc_s\cos\delta - d_{as}) + F_x(lc_s\sin\delta) \end{bmatrix}$$

where:

  - l is the distance between the leading edge of the planes and the system metacenter
  - b_s is the wingspan
  - c_s is the chord
  - S_s is the surface wing defined by $S_s = b_s c_s$
  - l=0.2 for the taipan class of vehicles.

B. Sensors modeling

The proprioceptive sensors are used to measure the state variables of the robot, and its derivative. These sensors are modeled here by using directly the variables produced by the vehicles simulator. Afterwards, the sensor simulator provides the samples at the same frequency as the real sensor, taking care to limit its range and adjust its resolution. It is also possible to add noise so as to obtain a more realistic simulation. Presently a GPS (Trimble Lassen SKII), a loch doppler (RDI Workhorse Navigator Doppler Velocity Log), as well as an attitude and heading reference system (XSens MTi) are modeled. As the physical phenomena driving the exteroceptive measurements are complex, the modeling process of the exteroceptive sensors can be a non trivial task. We are presently working on a simple sonar model using classic ray tracing method.

C. Environment modeling

The environment is modeled using different elements: the topography, the temperature and salinity distribution, the environmental disturbances. These informations are included in a single function $[temperature, salinity, current] = f(x, y, z)$. Presently, only the topography, the temperature and salinity distribution are implemented. This model considers these phenomena as stationary. More details about the architecture of *Thetis* can be found in [10].

IV. Control architecture of the robot

Such a simulator is fully useful only if the connection with the robot and its control architecture does not require any

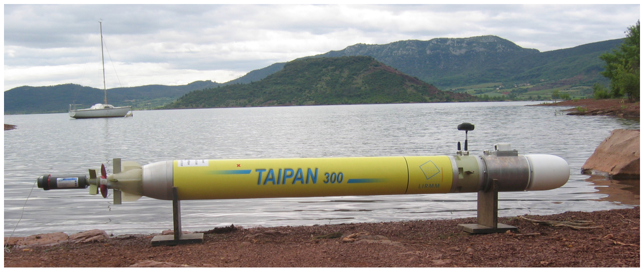

Fig. 5. Taipan 300 in front of the Salagou Lake in France

modification on the control architecture. Indeed, the transition from simulation to real experiments has to be as transparent as possible, unless the expected behavior wont be guaranteed. Thus the best way is, of course, to physically connect the AUV sensor devices to the output of the sensors simulator.
In order to do this, the sensors simulator must be able to reproduce the electrical signals of each sensor. But this solution is very difficult to implement. Indeed, often, constructors dont provide complete information about the sensors internal specifications. Hence, another solution have to be implemented, for which the code modification has to be as light as possible, in order to avoid inducing specific behavior of the control architecture. Thus, from the robot's controller side, it is necessary to implement a modular architecture, in order to modify only the data supply mechanism of the control architecture, shunting the real sensors.
For its part, the sensors simulator has to provide the sensors data, with the same updating rate, range, errors etc... as the real ones. In this context, we stay close enough to reality, in order to validate our control architecture.

A. *Control architecture of Taipan 300*

We briefly present here the architecture of our robot Taipan 300 which has been developed by our team. Taipan 300 is a small AUV which is designed for shallow water operations. It is 193cm long for a diameter of 15cm and a weight of 32 Kg (fig. 5). Its control architecture is a mixed architecture composed of 2 levels: a decisional level containing a global supervisor and several local supervisors (one for each mode: autonomous, teleoperation, and cooperation) and a executive level based on a set of modules in charge of low level control and instrumentation management, all these modules being under the control of a scheduler. Three types of objects are used in this architecture. The global supervisor receives from the operator a set of *objectives* which defines the mission to be achieved. Then it transmits a sequence of *objectives* to the concerned local supervisor; the latter sends *sub objectives* to the scheduler.

1) Scheduler: Two types of instruments can be found on Taipan 300: actuators and sensors. The first ones are managed by Perception Modules (PM) and the seconds by Action Modules (AM).
A PM is constructed for each data type (called variable) required by the architecture (e.g x, y, z, θ, φ, ψ). It is often

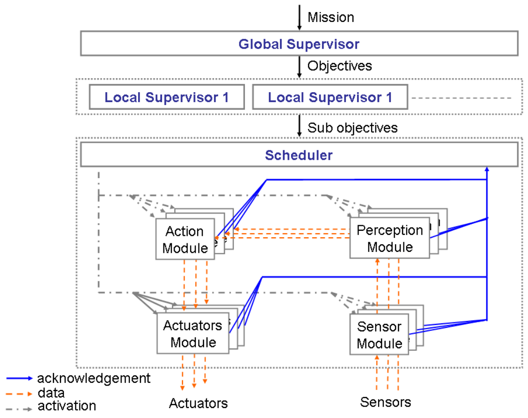

Fig. 6. Control Architecture of Taipan 300

necessary to use data coming from different sensors in order to compute or estimate a variable. Thus a PM is able to manage several sensors.
PM are in charge of 2 types of computation. The first one is relative to the operating mode of the sensor, and the second one is relative to the data computation of the sensors.
The AM which contains the command laws, allows us to compute the command to be sent to the actuators.
All these modules are activated and configured by the scheduler. Thus in order to get the position of the AUV, the scheduler sends an order to the corresponding PM and to reach a position, it sends a order to the AM managing thruster and fins. PM and AM work in a periodic way. This architecture is represented on figure 6.

2) local supervisor: A Local Supervisor (LS) is dedicated to manage a resource in a given mode. Concerning Taipan 300, we only have one ressource (the AUV) which has 3 operating modes *Autonomous Mode* (objectives from the GS are executed), *Teleoperation Mode* (low level teleoperation of the vehicle by the end user) and *Cooperation Mode* (commands the AUV in order to position the vehicle in a flotilla). We only focus here on the *autonomous mode*. The LS receives objectives from the GS to be executed. In order to do this, the scheduler decomposes the objectives into sub objectives and schedules the different modules (Perception and Action).

3) Global supervisor: The Global Supervisor (GS) receives from the operator (or from a mission handler) a file containing one mission to be achieved. A mission is described as a succession of *objectives* which must be reached by the system during the mission. The objectives can be a displacement or other actions to be achieved in a given geographic place (e.g. a bathymetric mission). These objectives can be reached successively or in parallel according to their nature. Information about their planning is given by the end user before the departure. It allows the GS to sequence correctly the objectives which constitute the mission. The GS also checks up that the objectives which are executed at the same time don't use the same resource. To reach an objective, the AUV passes through several steps. These different steps require the

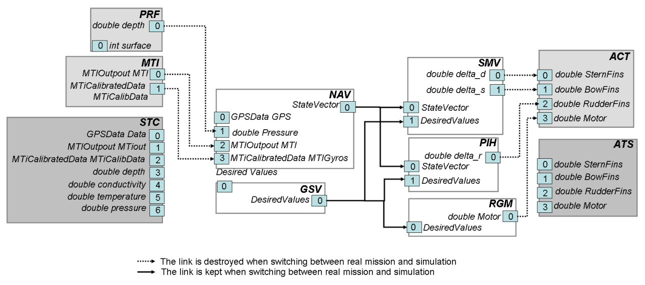

Fig. 7. Connection between the architecture modules

use of different command laws and instrumentations. Thus each objective is decomposed in several more simple entities. Only one command law is needed by these entities, which are called sub objectives, during their execution time. For example the objective *inspection of a pipeline* will be broken up into several sequences *diving, going to, tracking pipeline, following pipeline*. Finally, the GS sends to the concerned local supervisor the objectives to be achieved. The latter sends an execution report after each sub objective is reached. More details of this architecture can be found in [11] and in [12].

V. Connection of the systems

As the architecture presented above respects the required modularity to connect the AUV to the simulator, we are able to validate the architecture of the AUV. The behavior between real and simulated mission should be the same. In order to do this, we have to design the interactions between the modules. In order to program a mission, we need to interconnect the necessary modules. This step of the mission conception is preceded by different processes (precedence graph, required ressources etc...). Due to space limitation, the presentation of these processes will be kept for a future paper.

So in order to program a simple setpoint mission (keeping a desired heading for a given duration), we have to interconnect the modules as shown on fig. 7. The linkage of modules consists in establishing dynamic or static links which support the data and control flow upon the architecture. A module carries 6 categories of ports (data input port, data output port, events input port, events output port, parametrization port, request input port). The activity of these modules are controlled by a particular module called *scheduler*. On figure 7, light gray block represent the modules used during the real mission. The dark gray ones represent those which are used during the simulation. Finally the white modules are shared modules used during real missions and simulations. As we can see, we only have to replace the light gray module by the dark grey ones to switch between real mission and simulation. On the left side *PRF* and *MTI* modules are respectively the modules which are in charge of managing the pressure sensor and the attitude and heading reference system. These sensor modules are replaced by the *STC* module when simulator is used. This *STC* module is in charge of communicating and decoding frames from the simulator. It produces data for the other modules. Then several white modules are used in order to achieve the mission. We dont give further details here. At the end of the jobstring, the computed commands are reachable by the actuators module ACT. For simulation purpose, they are directed to the *ATS* module which is in charge of communicating with the simulator in order to provide the computed commands. In this way we warrant that the behavior of the architecture is almost not affected by the use of the simulator. Thus an hybrid simulator like *Thetis* is fully operational taking its interest with such an architecture.

Thanks to mission logs we can verify that the execution of the modules sequence is the same in both simulation and real mission.

VI. Conclusion

In this paper we have seen that HIL simulators with an adequate control architecture could play an important role in the development of robots controller. *Thetis* is a real time multi-vehicles simulator which need to be completed (environment simulator) but which is already exploited in order to validate our command laws. The relevance of the results comes, on the one hand, from upgradability, modularity and portability of the hybrid architecture of *Thetis*, and on the other hand, from the modularity, the scalability and the robustness of the mixed architecture of the AUVs controller. We have only evoked the use of this simulator in a single-robot underwater frame but it is generic enough to be used with other robots and thus we can imagine more complex scenarii like the coordination of AUVs and air drones in order to make coastal monitoring.

References

[1] P. Ridao, E. Batlle, D. Ribas, M. Carreras *Neptune: a hil simulator for multiple UUVs*, OCEANS '04. MTS/IEEE TECHNO-OCEAN '04, 2004

[2] T. Perez, ØN. Smogeli, T.I. Fossen, A.J. Sørensen, *An Overview of the Marine Systems Simulator (MSS): A Simulink Toolbox for Marine Control Systems*, Modeling, Identification and Control, 2006

[3] D. Suriano and C. Moriconi, textitA Distributed Simulator for the Development of the Unmanned Underwater Vehicles Control Software, Robotics and Applications and Telematics, 2007

[4] J.M. Spiewak, B. Jouvencel, P. Fraisse, *A New Design of AUV for Shallow Water Applications: H160*, ISOPE'06: International Offshore and Polar Engineering, 2006

[5] TI. Fossen, *Marine Control Systems: Guidance Navigation and Control of Ships, Rigs and Underwater Vehicles*, Marine Cybernetics AS, 2002

[6] LL. Whitcomb, DR. Yoerger, *Development, Comparison and preliminary experimental validation of nonlinear dynamic thruster models*, IEEE journal of oceanic engineering, 1999

[7] M. Aucher, *Dynamique des sous-marins*, Sciences et techniques de l'armement, 1981

[8] SA. Santos, *Contribution à la conception des sous-marins autonomes : Architecture des actionneurs, architecture des capteurs d'altitude, et commandes référencées capteurs*, Thèse de l'Ecole nationale supérieure des mines de Paris, 1995

[9] P. Ridao, M. Carreras, J. Batlle, J. Ama, *A New Hybrid Control for low cost AUV*,Proc. of the Control Application in Marine Systems, 2001

[10] O.Parodi, A. El Jalaoui, B. Jouvencel, *Thetis A Multi-Vehicule Hybride Simulator*, Proc. of the 17th IFAC congress, 2008

[11] A. El Jalaoui, D. Andreu, B. Jouvencel, *Contextual Management of Tasks and Instrumentation within an AUV control software architecture*, Conf. on Intelligent Robots and Systems (Iros06), 2006

[12] A. El Jalaoui, D. Andreu, B. Jouvencel, *AUV Control Architecture for Control Management of Embedded Instrumentation*, 4th IFAC Symposium on Mechatronic Systems, 2006

[13] *http://sourceforge.net/projects/doxygen/*, accessed on 17/10/2007