HNG: a Hybrid Network-based Generic architecture for fast prototyping and evaluation of robust autonomous vehicle systems.

R. Jaulmes, E. Moliné

DGA - CEP
16 bis, avenue Prieur de la Côte d'Or
94114 Arcueil Cedex

Abstract

The robotic researchers all agree that control architectures should be deliberative, reactive, robust, modular, and multi-robot. The HARPIC architecture that was developed with these ideas by GIP to show modular autonomy functionalities, had an" attention" feature that allowed the architecture to automatically switch from one algorithm to another according to the circumstances. Using the return we had about HARPIC, we design a new architecture, the Hybrid Network-based Generic architecture, based on concepts developed by different teams: it uses independent processes as modules, communication uses Inter Process Communication, and the "attention" feature is realized through a protocol inspired from Contract-Net: this protocol builds customer / supplier relationship within the processes and allows comparisons between them. Preliminary tests we have conducted on this architecture project prove the robustness and adaptability of this method. HNG is also a robust framework to benchmark the advanced techniques used in autonomous systems.

Keywords

Robotics ; Control architecture ; Real Applications ; Robustness ; Reliability ; Modularity ; IPC ; Multi-agent ; Contract-net Protocol.

1 INTRODUCTION

One of the main challenges in autonomous mobile robotic is the conception of mobile systems able to quickly and efficiently answer their users' needs. They must easily integrate the best and latest algorithms from a wide set of domains such as control, image processing, data fusing or AI for instance. They also must be easily adaptable to the new emerging technologies or new concepts. One of the main issues, in the conception of a control architecture, is to take into account and to combine as well as possible all this different abilities.

One can number a lot of various control architecture [1]. In a way, one can say that every laboratories may have developed is own that have been perfectly designed to answer their issues**.** Therefore this kind of architecture cannot easily be used for others applications. Moreover, because theses architectures have been developed independently from each others, it is often impossible to make interaction between them. Bringing one algorithm from one architecture to another, without decreasing its performance is a complex task requiring a perfect knowledge of both architectures and algorithm.

For a decade**,** more and more projects address the standardization issue [2][3]: they are gathering a increasing libraries of algorithms but are still in concurrence especially because they are not compatible and they are not offering the same functionalities. In our evaluation activities and capitalization of robotic technologies [5], getting standardized architecture to ease the integration has become a necessity. Without any will of standardization but taking into account the possibility of an integration of new algorithms as easy as possible, we have developed the HARPIC architecture [6] in order to validate two new operational concepts: the adjustable autonomy, which allows the user to give the robot commands of different levels, and the attention function, which allows a comparison between similar perception algorithms. As we want to ease the integration of new algorithms and new features, we have in a first time formalized our needs. In a second time we have studied the possibility to integrate the desired new mechanisms to an already existing architecture. We have also studied the possibility to extend the mechanisms of control developed during the ACROBATE project [7]. All these studies have allowed us to prototype a new decentralized control mechanism.

This article presents our works on the definition and the conception of the HNG architecture (Hybrid Network-based Generic Architecture) which is based on the results of our reflection and on some principles of already existing architecture. Part 2 and 3 present the analysis of our need about the architecture we wish to use in our evaluation activity and some existing architectures we have considered. In Part 4 we try to formalize the issue the controller has to solve before describing HNG concept in part 5. Finally part 6 presents some preliminary results that let us think that HNG would be a useful framework in our robotic activity.

2 ANALYSIS OF THE NEED

The first step in our study for the conception of HNG is to identify and analyze the qualities we want the architecture to have. The main objective is to use HNG to evaluate the performance of various algorithms in operational conditions and to study their robustness. In order to fulfill this objective the HNG architecture must have some qualities: robustness, security, modularity, to ease maintenance, adaptability to a multi-robot utilization, adjustable autonomy, and self-reconfiguration. Of course, it is obvious that in a first time, HNG should have all the necessary interfaces with robotic platforms, their sensors and actuators such as interfaces with the user. Moreover HNG must have a maximum of functionality that can contribute to the global autonomy of the systems.

Here are described the main qualities we want to provide to HNG:

- *Robustness*: one of the most primordial aspects in robotic control architecture is the robustness to the execution failure. All must be done in the architecture to avoid the system stop working in the correct way. Whatever the circumstances, the system must be as *fault-tolerant* as possible. As said before, we focus on the evaluation of algorithms coming from various domains, and from various sources. We often can not master all the code sources. The addition of a new algorithmic module (or functional module) should not make the architecture become unstable. In others words, the failure of a part of the system, e.g. of one or more modules, should not be synonymous with the failure of all the architecture, whatever the nature of this failure (lack of memory, data reading mistake, segmentation fault, …)

- *Security*: the notion of security must be taken into account in HNG. The integrity of the materials and the security of the persons using it are very important. Security is a constant and top priority task in the system. Therefore the architecture should use

parallel mechanisms (threads, multi processes) and be able to manage priority between these mechanisms.

- *Modularity*: we wish the addition of new functionalities to the architecture can easily use the existing algorithms. One of the best ways to answer this need is to divide the architecture in modules. Each module is a part of the architecture that can change, upgrade, or evolve all the time during the projects. To separate the architecture in independent modules also allow the distribution of the architecture on several calculators (GUI PC, onboard PC, fixed PC …).

- *Maintenance*: the architecture must be designed to ease the maintenance. Especially the reconfiguration and the re-launch of a module must be possible while the system is running and without interrupting the experiments. Moreover the module must be able to record and save online their internal data and their interfaces so that in case of failure, it is possible to identify the module(s) responsible of this dysfunctional execution. The modules can also be tested alone, their input and their output perfectly controlled.

- *Multi-robots*: the scientific community is working more and more with the issue of the cooperation between robots: the HNG architecture must intrinsically be fitted with the conception and the use of multi-robots systems. The modules of the architecture can be located on several computers or robots and the share of the information between the different platforms must be managed by the architecture.

- *Simulation*: the architecture should have simulation tools and interface. Simulation is a way to test the behaviors of the modules in perfectly well-controlled environments and therefore to evaluate in some conditions their performances. Hybrid simulations where a part of the data is simulated while the other part is real data could also be conducted.

- *Hybrid*: a part of the modules must come from the automatic domain and allow the control of the robot in a reactive way. But the use of these short close loops modules must not forbid the use of high level module allowing the planning of the activities of the robots by the use of models / representations of the world. The architecture must be hybrid.

- *GUI and adjustable autonomy*: the interface between the user and the system must give the user the ability to interact with the different level of autonomy of the robot. Therefore the system can both deals with high-level mission (exploration mission for instance) and with low-level control (remote control for instance). The GUI is also a way to provide modules with configuration data and designation data that the user is the only one to provide. The GUI must be adaptable to both the needs of the user and the abilities of the system.

- *Self-reconfiguration*: this ability is very important. First, in case of failure of one or more modules, or when the chosen modules are no more able to fulfill the designed task, the system must self-adapt and find a new module or series of modules to efficiently do the task. Second, the architecture must fit the needs of the users and adapt itself to his change (from a full remote control interaction to a supervised remote control one for instance). The architecture must also update and change the data exchange between the modules depending of the circumstance.

3 HNG AND THE EXISTING CONTROL ARCHITECTURE

Here are some existing control architectures we have considered in order to check how they can answer to our needs. Our reflection is based on the analysis by Kramer and al. [1] of the robotic development environments for autonomous mobile robots. HNG has found its inspiration in some of the existing architectures, starting with HARPIC.

HARPIC [6]*:* it is a hybrid architecture based on posix threads and divided in two parts, the Perception, supervised by an Attention agent, and the Action, supervised by the Selection agent, and offering to the user a control interface allowing him to select the desired behaviors. The main particularity of HARPIC is that these two parts have self-reconfiguration functionalities depending on the self-evaluation of the algorithms for the Perception part and on the needs of the user for the Action part. Despite some works to open the architecture to the multirobots [8], to add new control and high-level decision mechanisms with the ACROBATE project [7], or non Action-Perception dividable mechanism such as the SLAM [9], this architecture is not perfectly answering our need, especially in terms of robustness. Nevertheless, the innovative concepts the HARPIC architecture has proposed are very interesting. First, the *adjustable autonomy* gives the user the possibility to send order of different level of complexity. Those can be for instance remote control orders, waypoints navigation orders, or image designed objects tracking orders. Second, the *Attention Agent* has to detect the agents that are the most efficient to answer the queries of the users regarding the environment and the events happening by analysis the representations of the world. It can also send at real time some functionality or action the system is able to execute. For example, if the *Attention agent* detects that a Wall can be used by the couple "*Wall detection Perception agent/ Wall Following Action agent"* this behavior is indicated as available trough the GUI.

Player/Stage [2]: Player/Stage is using two processes; one is either the simulator or the robot controller and the other is gathering all the control mechanisms (based on posix threads). HARPIC is using an identical mechanism to interface with the simulator or with the robot. HARPIC is also compatible with Stage (2D) and Gazebo (3D) simulators.

Carmen [3]: it is more a toolbox than an architecture. Here every module is an independent process so that the maintenance is easier to do. This toolbox contains control mechanisms, SLAM mechanisms, an inter process communication tools (IPC) based on the Publish/ Subscribe paradigm, interfaces with sensors and actuators, diagnosis and maintenance tools, such as a "watchdog" mechanism which allows to check during the execution the states of the different processes (and therefore of the different modules) and to rerun them if needed. Carmen has focused our attention because the first robots winners of the DARPA Grand Challenge used it or was inspired by it [10]. Carmen has been initially designed to run on several calculators and can be adapted to the multi-robot. Carmen is rather singular thanks to its robustness and maintenance mechanisms.

We have been convinced by Carmen and we have envisaged improving it by making the concepts of HARPIC compatible with Carmen. Nevertheless, not only HARPIC is written in C++ and Carmen in C, but also Carmen, by its lack of high level control mechanism or self-reconfiguration is not perfectly answering our needs. Therefore we have decided to develop our own control mechanism. We have decided to base our work on the Gazebo simulator of the Player/Stage project, on the new concept proposed by HARPIC, and on the communication and maintenance mechanisms of CARMEN. This new architecture called HNG should be as generic as possible in order to add others control principles from different architectures [1].

4 DEFINITION OF THE CONTROLLER

In this part we present the study we have conducted for the conception of the controller of the architecture. The controller has to decide which modules to activate and what are the data exchanges that must be done. We present a formal description of the issue that the controller has to solve, before we propose a solution of what the controller could be.

4.1 Formal description of the issue

The architecture is made of N modules $\{m_1,\ldots,m_N\}$. Each of these modules can be activated or deactivated, and has a set of inputs, for the data it consumes, and a set of outputs for the data it produces. Every data is typed so that some input/output connections are not valid. Let **C** the matrix, unique for all the system, taking its values in {0;1} and indicating the existence or not of a connection between the different inputs/outputs. **C**(*i*,*j*) = 1 if and only if the input *i* is linked to the output *j*. The controller must be able to decide at every time and among all the available modules, which are the ones to be activated. It also has to determine the coefficients of the matrix **C**. Several inputs can be connected to a same output. It means that the published message is sent to several subscribers. In a same way, several outputs can be connected to a same input. It is especially the case when different modules want to control the mobility of the robot. Therefore, priority rules must be defined in order to prevent conflicts. Theses actions of control have an impact on the consumed resources, the abilities to run and the qualities of this running.

The different modules m_i are also able to evaluate online a value $f_{mi}(t) \in \{0;1\}$ which indicate their aptitude to function, to do their task. When this value is 1, they also calculate another value reflecting their working quality $Q_{mi}(t)$ whose values are in [-1;+1]. $Q_{mi}(t)$ can be biased. If a module cannot evaluate its working quality, the values of $Q_{mi}(t)$ will be 0.

Finally they are able to evaluate the amount of resources they are using. Let $\mathbf{R}_{mi}(t)$ a vector of size Z, where Z is the total number of the different resources available in the system : CPU and memory resources (RAM)of each computer, and the density of the communication in the network.

One of the modules is called the Manager. The control mechanism must verify that this module can be activated and that its working quality is maximal. The global resources of the system are limited and the controller has to check that the inequality (1) is always true. ($\mathbf{R}_{max}$ is a vector characterizing the maximal amount of resources the system is able to supply).

$$\forall t, i, \qquad \sum R_{m_i}(t) \leq R_{\max} \qquad (1)$$

The controller has the knowledge of the mission. This mission which can be modify online depending on the will of the user is constituted by a set of objectives $O_1(t),...,O_P(t)$, respectively associated with a hierarchical priority. The priority of the objective $O_i(t)$ is higher than the priority of the objective $O_j(t)$ if and only if $i > j$. For each objective O_k, the controller knows the set $\Gamma(O_k)$ which contains the modules which can answer this objective. To complete the mission, the Manager has to find the best modules which can fulfil these objectives thanks to the service they provide. In the next section of this paper, we will use without any distinction the terms objectives or service.

The controller has to guarantee the execution of a maximal number of the objectives of the current mission while guaranteeing the hierarchy of the objectives at each time. In others words, the rule (2) must always be true.

$$\forall i,j,t \quad i>j \Rightarrow \left[\exists m_x \in \Gamma(O_j(t))\ f_{m_x}(t)=1 \Rightarrow \exists m_y \in \Gamma(O_i(t))\ f_{m_y}(t)=1\right] \qquad (2)$$

One interpretation of this rule is that as long as the resources are available, all the objectives have to be fulfilled. In a case of lack of resources, the controller has to check that the choice between two objectives will be in favor of the one with the highest priority.

The issue could be present as an optimization under constraints, applied in an environment where both the target function and the constraints vary with the time. Moreover, because the controller is the controller of an onboard system, the decision must be done in limited time and if possible, independently of the number of modules to manage.

It is impossible to solve in a precise way this problem because it would mean to know very precisely the needs and the working domains of each module. Therefore the methods based on the linear programming [11] are not interesting. Others methods based on supervised learning (SVM, neurons network…) could be envisaged but we have chosen to not use them because the leaning should have to be done each time a module is modified. About methods based on probabilistic models such as Bayesian network [14] it would be difficult to catch with enough precision all the phenomena that can happen for a set of a given module. Moreover, the adjustment of the parameters of these models is not an easy task to do, and they could change each time a module is modified. The weakness of all this control mechanisms is that they are centralized. The centralization is needed to deal with the conflicts created by the use of the shared resources. To detect and manage these conflicts must be done in a centralized way but the optimization of the working quality could be done in a decentralized way. Indeed, with the assumption that the modules are able to determine what they need to do their job and that they are also able to estimate their working quality, thus they are able to decide which connections to create with which modules.

4.2 Description of the proposed solution

Consider the following assumptions: each module is associated to a vector of needs **N** (the services he is waiting) and to a vector of objectives it can fulfill **K** (the services he is able to provide if all its needs are satisfied). Each of these needs/objectives can be linked to inputs/outputs of type T.

- If for each need n_i of the vector **N** of a module m, there is another module m' such as $f_{m'}(t) = 1$ and its vector **K** contains the service n_i (so it is said that m' can answer the need n_i of m) then $f_m(t) = 1$.
- The value of $Q_m(t)$ increases if the value of $Q_{m'}(t)$ increases, m' being one of the modules answering one or more needs of the module m.
- Each need of the module m has a unique level of priority. This priority is proper to the module m. The needs with the lowest priorities are not indispensable to obtain the equality $f_m(t) = 1$ but they can have an effect on the value of the working quality of the module.

We use theses assumptions to define a protocol that guarantees to have, if she exists, the best working quality under constraints (the available resources of the system). This mechanism is close to the Contract-Net protocol [12] which is rather well known in the multi-agent literature.

The Manager module in order to fulfill its objectives (and thus the mission) has to send queries. Some modules will be able to answer this queries and thus to establish a contract with the Manager module. But before that, they have to express some queries too and make some contracts with others modules, and so on until all the needs are fulfilled (or until there is no module able to answer the needs)

Because several modules can answer the same queries i.e. they provide the same kind of service a mechanism has been defined to help the Client module to choose the best Provider. This mechanism is used to note the efficiency of each candidate. Each module and thus each Provider has a value function $V_{mi}(t)$ which is linked to its self-evaluation (working quality) but not only. $V_{mi}(t)$ must depend on the effective consumption, by the module m_i, of the resources $R_{m_i}^{Eff}(t)$, whose the description is written in section 5.3, and also on the rarity of the resources through a centralized cost of resources $\mathbf{C}_R(t)$ whose the coefficients are regularly updated.

$V_{mi}(t)$ must also take into account the cases where the self-evaluation is not efficient. As a module is able to detect, when it measures a decrease of its quality, which modules among its provider are responsible of this, it could be interesting to modify the values of a module by the feedback of the clients of this module. Let $E(m_i,t)$ the set of the evaluations transmitted by the client of m_i between the time 0 et the time t, let E_n one of these evaluations (whose the values is between -1 and 1), and $T(E_n)$ the time when this evaluation has occurred, and finally let $\tau(m_i)$ the characteristic length of time for the validity of these evaluations. The principle used to evaluate the value of a module is summarized in the following equation:

$$V_{m_i}(t) = Q_{m_i}(t) - R_{m_i}^{Eff}(t).C_R(t) + \sum_{E_n \in E(m_i,t)} E_n e^{-\frac{t-T(E_n)}{\tau_{m_i}}} \qquad (3)$$

The value of a module increase with a high value of its working quality, a low consumption of resources and a high evaluation feedback from its clients. One can notice that the last term of (3) gives more weight to the latest evaluations. The feedback evaluation by the client can also 'correct' a biased self-evaluation. It can be used by the client to force a change between a bad but optimistic provider (A) and another better but pessimistic provider (B). Indeed, as long as (A) will provide its services to the client module, it will receive bad feedback evaluation, thus $V_A(t)$ will decrease and when $V_B(t)$ will become higher than $V_A(t)$ plus a threshold, the client module will change its provider. As (B) is good, it should receive a good feedback evaluation from the client. Even if the value of (A) will progressively increase, she should stay inferior to the value of (B). We advocate that the evaluation sent by a module to its providers is linked with the evaluation this module has received from its clients. If we do so, then if the Manager module is able of a non biased evaluation the system can be balanced.

5 DESCRIPTION OF THE HNG ARCHITECTURE

We now describe the general principle of HNG, the communication mechanism, the resources managers, the failures management mechanism and the generic mechanism used by all the modules. There is also a list of module implemented in the 1.0 version and a description of the architecture on an example.

5.1 General principle

HNG uses the principles and the communications methods of CARMEN combined with the control mechanism presented in section 4. On the software point of view, all the modules are independent processes.

As Carmen, HNG has « Central » processes (no more than one by calculator) whose the function is to transmit information inside the architecture: each module of HNG is connected to at least one Central and he can receive messages from all the Centrals he is connected to. HNG also have a one administrator process for each calculator: these processes are called "Resources Manager" (RM). They receive different activity reports from the modules running on the calculator they have to manage, they calculate and transmit the global cost of the resources $C_R(t)$ we mentioned in 4.2, and they also calculate for each module le effective resources cost vector $R_{m_i}^{Eff}(t)$. The RM can also stop (kill) and re-start modules if they have made a failure in their execution. At the beginning, when a RM process is starting, a set of modules is also started and configure regarding the values of parameters of the configuration files. In this file, one of the modules can be designed as the Manager module. This module will have the priority with the highest value.

Having a set of processes rather than a set of threads is more advantageous for several reasons:

- The architecture can run on several calculators, which is interesting for multirobots applications.
- The operating system can be easily used to determine precisely and easily the amount of memory and computation load used by the modules
- If one of the modules (thus a process) makes an execution failure (as a segmentation fault) there is no influence on the others modules. The architecture is more stable to unexpected failure of some modules.
- We gain time when doing diagnosis of failures, because no memory is ever shared between the modules and therefore determining which module failed when a crash occurs is instantaneous.

5.2 The Communications

The communication in HNG is based on the IPC library developed by CMU [13]. This library of inter process communication functions is based on TCP Unix sockets communications and allows the transmission of data by Ethernet (between two robots of two computers, or one computer and one robot). IPC offers mechanisms to interrupt a process at the time of the reception of a message. That means that a process is not wasting time waiting a message and checking an empty mail box. IPC also allows the management of periodic events and is compatible with the multi-thread.

IPC offers Publish/Subscribe functionalities working on the following principle: a module can subscribe by indicating to IPC Central the name of the Channel he is interested in, the structure that the messages using this Channel must have, and what kind of functions (handlers) to call at the reception of one of these messages. The module can also send message by indicating the name of the Channel he want to use to emit its message. IPC knows the identity of the processes which are interested in this message and he automatically send a copy of it: the processes execute the predefined function (handler) at the reception. IPC also dates the send messages and provides a common time reference for the system. It is also possible to define and use several Central in order to avoid saturation in transmission. All the different mechanisms of communication have been encapsulated in a unique C++ class.

5.3 The Resources Managers

There are three kinds of resources in HNG: CPU resources, memory, and the communication load of the Centrals. Each Resources Manager has to calculate the effective consumption of resources of each module presents on the same calculator and to estimate the remaining resources. RM is using the information given by the operating system (OS), the information periodically send by the modules and the information send by the others RM of the others calculators of the system.

The communication load of a Central is measured by the RM present on the same calculator by measuring the time differential between the moment the reports message were sent and the time of the reception of this message by the RM. If this time is higher than a defined threshold, the cost of the communication is increasing. Otherwise this cost decreases until a minimal cost.

For each module m_i managed by the RM, and a each instant, its CPU and memory resources consumption $\mathbf{R}(m_i) = R_{m_i}(t)$ is measured using tools provided by the OS. Then the RM can calculate measures the effective resources consumption of each module. It built the set $F(m_i)$ which contains the indices of the different modules chosen as providers to answer the needs **N** of m_i. It also determines the number of queries $N_R(m_j)$ answered by a module m_j. The effective cost $R^{Eff}(m_i) = R^{Eff}_{m_i}(t)$ used in the calculation of the value $V_{mi}(t)$ is calculated using the formula (4).

$$R^{Eff}(m_i) = R(m_i) + \sum_{j \in F(m_i)} \frac{R^{Eff}(m_j)}{N_R(m_j)} \qquad (4)$$

The more a module answers queries, the less its weight is important on the effective cost of its clients. In others words even if a module consumes a lot of resources, the more he has clients the less it impacts their values.

Another function that a RM must be able to do is the management of the resources in situations of lack of resources. If a module needs to consume resources to do its task and if there is no more resources available he will put in its report for the RM, "waiting for resources to answer queries X". The RM measures the amount of available resources in term of memory and CPU time. If the quantity falls under a fixed threshold, the RM will increase the cost of the corresponding resource and will update the cost vector $\mathbf{C}_R(t)$ seen in 4.2. Otherwise, it will decrease the cost of this resource. As soon as the RM receives at least one report including information of lack of resources, it will increase their cost. If despite a maximal cost for the resources, it still receives this kind of reports, it can activate the "degraded mode". In this case, a message indicating the minimum priority a query must have to be taken into account despite the lack of resources is sent to all the modules under its responsibility (it means on the same calculator of the RM). The modules providing the services useful for the query with a priority lower than the minimum one are not allowed anymore to work. The definition of the priority between two queries is defined in 5.7.

The modules which are not allowed to answer queries because of the degraded mode are still sending reports message to their RM. As long as the RM receives lack of resources report messages or as long as it detects that the system have no more available resources, the degraded mode is active. This mechanism has been defined in order to assure that even if the system has not enough resources to fulfill all the objectives (even by using the modules consuming few resources), the queries the system will not answer will be the one with the

lowest priority. Of course, the more the lack of resources increases, the less the number of answerable queries is. This can even be synonym of a global stop of the system. But this mechanism guarantees that until the end, only the queries with the highest priorities are taken into account. The figure 1 presents the different interfaces of the RM. The failures management is described in 5.4

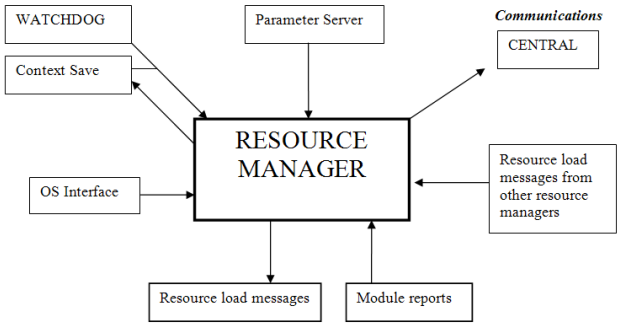

FIG. 1 – Interfaces of the Resource Manager.

5.4 Failures management

HNG has a mechanism for the management of the execution failures. Periodically each module writes in a file all the data concerning its state of running. If the process is stopped (killed) and restarted it can detect the presence of this file and thus it is able to come back at a previous state before the failure. If the failure occurs too often, the module is finally completely deactivated.

Every process p_i is associated with a "Watchdog" process which surveys it. The watchdog can detect an unusual end of p_i and therefore automatically and immediately restart it. The RM can do this Watchdog function for the modules he manages. Of course there is also another simple watchdog process for each RM. Each process also has a maximal authorized CPU load and memory load. The RM can then send interrupt signal to some module if needed (i.e. when they are too much resources consuming). The RM can also stop and restart a module which has not sent its periodic report, or which has sent a report indicating a too long time to provide the data.

5.5 The priority of the modules and the queries

We describe here how the absolute priority is defined. The priority is not linked to the nature of the query but it depends on the module asking this query, the position of this module in the contract tree and the internal hierarchy of the service this module have to request. Each module and each query in the architecture has an associated *priority:* as we have already mentioned, this feature is absolutely necessary when there is a conflict in resources (a module that can only process data from one client at a time, for instance). A priority is affected to each contract and is defined by a list of integers: $P_1.P_2.\ldots.P_i.\ldots.P_{k-1}$ where for each value i, P_i is an integer. We use the classic integer order to compare the priority. For instance the module m_1 with priority 1.2.5 has a bigger priority than the module m_2 with the priority 1.3.2.

Let R= $r_1, \ldots, r_n$ the contracts module m has signed with its providers (ordered by priority), and S = $s_1, \ldots, s_n$ the contracts module m has signed with its clients. Let's $P(s_i)$ be the priority of contract s_i. By definition the priorities of each of the contracts r_i is:

$$P(r_i) = (\max_{i=1...k} (P(s_i))) \cdot i.$$

The priority of the module m is directly linked to the priority of the queries is answer to. It is defined by:

$$P(m) = (\max_{i=1...k} (P(s_i)))$$

5.6 The generic mechanisms of the modules

We now present the standard mechanisms used by all the modules. By construction (inheritance of a generic class) all the modules can use these mechanisms and therefore they can assume each of these functions:

Producer: a module can produces data (representations, plans, commands…). He often produces its data on one (or more) channel he has created or on some channels he was asked to during the reply/query phases. These channels are named DEST1, DEST2 on the figure 2.

Consumer: a module can use and process data (representations, plans, commands). If he knows the name of the channel where to take the data, he can listen on it and process with the data at their reception. This data processing can be associated with the production of others data. The name of the channel used to receive the data is either unique (it is the case when the module can process only one source at a time) or is indicated in the query he answer to. These channels are named SRC1, SRC2 in the Figure 2.

Provider: a provider of services is able to publish on a channel corresponding to the provided service (Query A on the Figure): he indicates the provided service and its value $V_{mi}(t)$ calculated as presented in section 4.2. If the service is to provide data, the name of the channel where the data are sent is also indicated. In the standard mode, a module provides its service only if it has been selected as a provider for a query. This information is sent on the "Query A" channel in the Figure 2. If the module is the "Manager" module, he automatically provides its services as soon as they are available. The provider module receives on the channel "Query A" the feedback evaluations of its Clients modules. These evaluations are used to calculate its values $V_{mi}(t)$. They can also be used to improve its behavior: for instance he can indirectly transmit the evaluations he received on its own provider modules.

Client: a module is able to send a query on the "Query" channel corresponding to the needed service. He does so if he needs data or services to realize its work. The needed service could be a function, a data processing, or data. The query is sent on the "Query B" channel in the Figure 2. The module can know if a module is able to provide the requested service by listening to the "Reply B" channel. Once the client module has selected its provider, it's sending again its query but this time the message contains the selected module with its value, (used by the other providers to check if they are better or not than the selected one) and also the feedback evaluation if available.

Drivers: a module can also have some interface with the hardware like the sensors and the actuators. In this case, the RM has to control than one and only one module is the interlocutor of a specific device. To do this it uses the priority of the query. For instance if two Client modules have to control the position of a pan and tilt camera, the client with the highest priority query will be the only interlocutor of the camera Drivers module.

5.7 List of the modules of HNG 1.0

The modules of HNG v1.0 are listed in Table1. For each module, the table presents the given services (the queries the module can answer), the needed services (the queries the module must ask) and the inputs and outputs for the exchanged data. On this first version the listed modules concern the navigation of a mobile robot, but there is no limitation to define and implement others specific services like the "survey of an area" for instance.

All these modules are based on the different behaviors already implemented in HARPIC. The next step will be the implementation of the already realized works during the ACROBATE project [7] and of others functionality developed for the control architecture in freeware licenses.

Module	Services	Needs	Data in	Data out
Manager		Robot integrity Human control		
Security	Robot integrity	Robot control LIDAR data	LIDAR data	Commands
Interface	Human control Beacon designation	Robot control Follow wall Reach beacon Reach goal Localization Mapping Video Beacon tracking Wall detection	Map Position Images	Commands Goal Beacon
Robot	Robot control Odometry		Commands	Odometry
Simulator	Robot control Odometry LIDAR data Video		Commands	Odometry LIDAR data Images
Laser	LIDAR data			LIDAR data
Camera	Video			Images
Planning	Reach goal	Reach waypoint Mapping	Goal Map Position	Waypoint
Servo-controller	Reach waypoint	Robot control Localization	Waypoint Position	Commands
SLAM	Localization Mapping	Odometry LIDAR data	LIDAR data Odometry	Map Position
Beacon reacher	Reach beacon	Robot control Track beacon	Beacon	Commands
Visual tracker	Track beacon	Beacon Designation Video	Images Beacon	Beacon
Wall follower	Follow wall Detect wall	Robot control LIDAR data	LIDAR data	Commands
GPS/IMU	Localization			Position
Maps	Mapping			Map

TABLE. 1 – List of the HNG v1.0 modules.

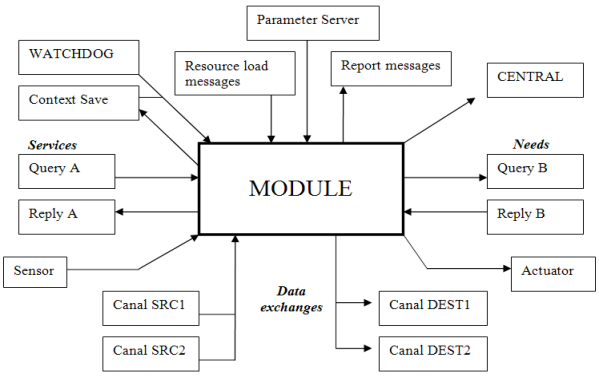

FIG. 2 – Interfaces of a generic module.

5.8 Example of application

In order to illustrate how HNG works, we describe in Figure 3 and 4 the mechanisms associated with the use of the modules described in 5.7. The contract tree (Figure3) shows the different required queries and established contract and therefore illustrates the Client/Provider relations between the modules. The data exchange figures (Figure 4) shows the real data exchange network between the modules and thus illustrates the Consumer/Producer relations. One has to keep in mind that these two diagrams are automatically and simultaneously created and that self-reconfiguration can occur.

In a first step, a module asks for its queries and gets the values of all the potential providers. In a second step, the module informs them of the selected one and sends the value $V_{mi.}$ The connections between the different active modules are created in order to allow the exchange of data, and the selected providers also have to inform which one of its sub-providers have been selected. The mechanisms is recursively executed until all the connections are created (or until no service can be provide)

Periodically, the values $V_{mi}(t)$ of the different modules are updated. The non-selected providers check if their values has became better or not than the value of the selected one. If so, a message is sent to the Client module which cancels the contract in order to establish a new one with this new module providing a better service. This mechanism therefore means a modification of the contract tree and a reconfiguration of the data exchange network.

In the proposed example, the queries send by the Manager module is first the security of the platform and then the control of the robot by the user. The user wants the robot to reach a goal point. The architecture finds that at this time, the SLAM module is more efficient than the GPS module and the SIG module.

About the priority, the two queries asked by the Manager have the priority 1 (Security query) and the priority 2 (Human Control query). Therefore the queries asked by the module answering the "Security" query will have a priority higher than those asked by the module answering the "Human Control" query. So the priority of "Control" query asked by the "Security module" (priority 1.1) will be higher than the priority of the "Control" query asked by the "GUI module" (priority 2.1). In the same way, one can determine that the priority of the "Control" query asked by the "Servo-Controller module" is 2.2.1.1. priorities. Therefore, the operator can contradict the commands of the autonomous navigation done by the "Servo-

Controller module", and both these commands can be invalidated by the "Security" component, that ensures the integrity of the robot.

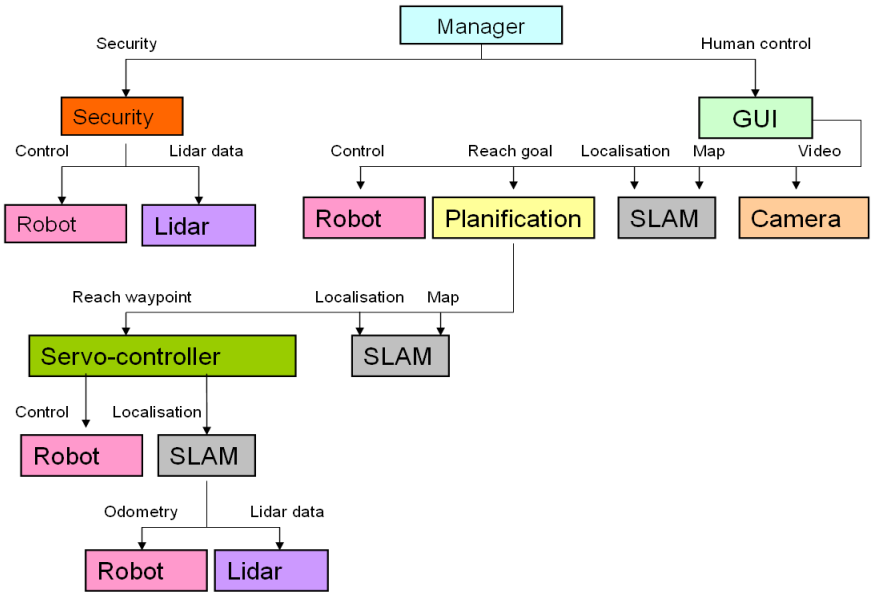

FIG. 3 – Example of a contract tree.

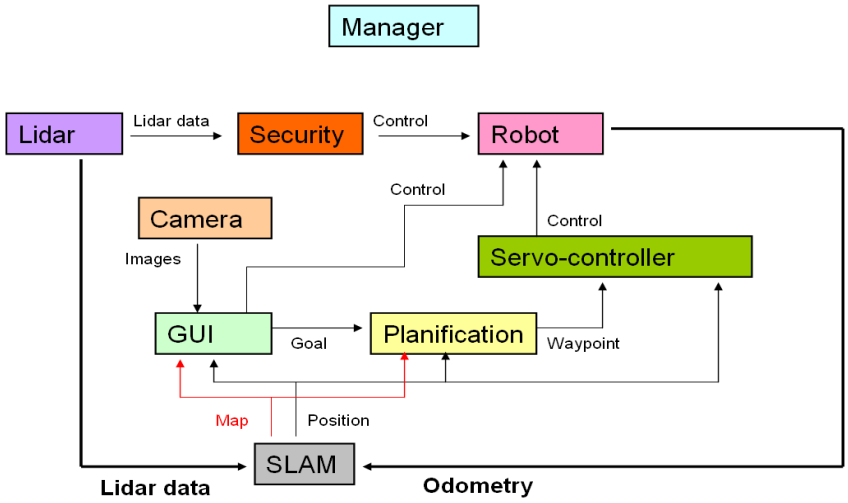

FIG. 4 – Example of a data exchange network.

6 PRELIMINARY RESULTS

The conception of the HNG architecture follows an incremental process. We validate the different mechanisms at each step of their realization. The following preliminary validation tests have been done:

- The encapsulation of the IPC communication protocol into an abstract class has been done. The modules are able to communicate in the same calculator and between two different calculators. The creation online of new channel is operational such as the dynamic redefinition of the structure of the message. We have measured the influence of

the volume of the transferred data on the time delay necessary to send the data. The Central is able to transmit 60 Ko by second with delay inferior to 20ms.

- The maintenance mechanism with the periodical creation of a file to save the context of a module and the load of these data in a restart process has been validated. On the same way the Watchdog mechanism use to detect unexpected failures and to restart deficient modules is correctly working.
- The Resources Managers are operational. They are able to access the OS information in order to measure the CPU time and memory consumption. They can also calculate the effective costs and to detect if a module has an unexpected consumption of resources and interrupt it if needed.
- The control mechanism has successfully fulfilled several simple tests in simulation. He is able to automatically build the contract tree and the data exchange network, and to self-reconfigure according to the update of the performance of the different modules varying with the change of the environmental conditions. When the auto-evaluation of a module is not representative of its real working quality, the system can partially correct them and force a reconfiguration.
- A lot of operational modules are available through CARMEN and the agents of HARPIC. Once these modules will be encapsulated in HNG, it would be easy to validate the concept HNG on real robotics systems in more realistic applications.

7 CONCLUSION

The bases of the HNG architecture have been defined, conceived and implemented. The needs/provider mechanism allows a self-reconfiguration of the system in case of an unexpected failure of one or more of the modules. The watchdog concept and the context save approach make the system more robust to execution failures. HNG would be an interesting framework for the development of a important library of modules, and the fact that both integration of new modules and maintenance of the systems have been thought to be as easy as possible will have a positive impact to ease the development of new modules and also to ease the implementation and the improvement of open source or off-the-shelf algorithms. HNG would become an indispensable tool in our evaluation activity of perception algorithms, sensors or concept of use for the robotic. Nevertheless the online self-reconfiguration has to be tested and validated on real robots with more operational modules and a quantitative evaluation of the quality of the reconfiguration mechanism has to be conducted. Our next objective is to extend the architecture to networks of combined ground sensors and UGV. We also want to study how the security aspect of the architecture can be formalized and how we can build proofs of reliability that would allow HNG to be used with UAVs. A study of the interaction between the robots and the operators and how HNG could adapt to them would be also interesting to conduct.

References

[1] Kramer J., Scheutz M., « Robotic Development Environments for Autonomous Mobile Robots: A Survey. », *Autonomous Robots*, 22(2):101-132, 2007.
[2] Gerkey, B. et al. "Most valuable Player: A robot device server for distributed control.", *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2001
[3] Montemerlo M., Roy N., Thrun S. « Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit », *IROS*, 2003.
[4] Jaulmes R., "Apprentissage actif dans les Processus Décisionnels de Markov Partiellement Observables", Revue d'Intelligence Artificielle, 1/2007.
[5] Lambert M., Jaulmes R., Godin A., Moliné E., Dufourd D., « A methodology for assessing robot autonomous functionalities », *IAV* 2007.
[6] Dalgalarrondo A., « Intégration de la fonction perception dans une architecture de contrôle de robot mobile autonome », thèse de doctorat, 2001.
[7] Lacroix S., Joyeux S., Lemaire T., Bosch S., Fabiani P., Tessier C., Bonnet O., Dufourd D., Moliné E. « Projet Acrobate, Algorithmes pour la coopération entre robots terrestres et aériens », *ROBEA* 2006
[8] Sellem P., « Navigation coopérative par échange de représentations de l'environnement », JJCR 2000.
[9] Dufourd D., Dalgalarrondo A., "Integrating human / robot interaction into robot control architectures for defense applications", *CAR 2006*
[10] Thrun S *et al.*, «Stanley: the robot that won the DARPA Grand Challenge», *Journal of Field Robotics* 23(9) pp. 661-692, 2006.
[11] Guéret C., Prins C., Sevaux M. , « Programmation Linéaire », 2000.
[12] Smith R., «The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver», *IEEE Transactions on Computers*, C-29, 12, pp. 1104-1113, 1980
[13] Simmons R., « Inter Process Communication library » http://www.cs.cmu.edu/~IPC/
[14] Naim P., Wuillemin P., Leray P., Pourret O., Becker A., « Les Réseaux Bayésiens », Eyrolles 2004