Inconsistencies Evaluation Mechanisms for an Hybrid Control Architecture with Adaptive Autonomy

Bastien Durand, Karen Godary-Dejean, Lionel Lapierre, Didier Crestani

Inconsistencies Evaluation Mechanisms for an Hybrid Control Architecture with adaptive autonomy

B. Durand, K. Godary, L. Lapierre and D. Crestani

LIRMM
161 rue Ada
34392 Montpellier Cedex 5
Université Montpellier 2 – CNRS

Abstract

This paper presents a set of mechanisms allowing the detection of control architecture inconsistencies, in the context of autonomous mobile robotics. This approach integrates an observation level into the structure of the control architecture to monitor and analyze the internal state of the robot and detect inconsistencies. These data are processed and the most pertinent information are sent to the Human supervisor and/or to a decision level which can adjust the robot autonomy according to the identified problem. These mechanisms have been implemented using architectural concepts previously developed at LIRMM. Based on these developments, adaptive hybrid control architecture is currently under development.

Keywords

Hybrid control architecture; mobile robot; inconsistency detection; adjustable/adaptive autonomy.

1 INTRODUCTION

For mobile robots, autonomy is one of the main objectives that the control architecture has to fulfill. However, autonomous robots cannot face all situations in an unknown and dynamic world. Humans can help the robots using their cognitive and acting capabilities to take decision. Human is an essential agent for complex robotic missions because he can overcome some of the control architecture limitations. Then, autonomy becomes adaptive and the design of the control architecture has to integrate the possibility of potential Human-Robot interactions, at different levels.

The first part of this paper presents the context of this study and recalls the main classes of control architectures. The concepts of autonomy, Human-Robot interaction and adjustable autonomy are then explained, and previous results related to fault detection are presented.

During the execution of its mission, an autonomous robot has to face potential dysfunctions, with different impacts on the system capabilities. These dysfunctions might not have any local solution (i.e. problem of battery power). However the problems come often from hardware failures (sensors, actuators, embedded electronic, etc.), software bugs or models limitations (approximation error, etc.). Usually these dysfunctions are not considered into the control architecture.

In the second part on this paper we detail a set of mechanisms allowing the detection of a predefined set of dysfunctions. These mechanisms allow for anticipating, detecting and processing a dysfunction. If the encountered problem cannot be solved autonomously by the

robot, the Human supervisor can adjust the robot autonomy, and he takes the necessary decisions in order to overcome the problem.

Then, we present how these observation mechanisms have been implemented using the architectural concepts developed in the LIRMM laboratory.

Finally before concluding and presenting some future works, the actual control architecture under development is exposed.

2   STATE OF ART

2.1   Classification of Control Architecture

2.1.1   Historical control architecture

Generally, the literature on control architecture proposes two types of control architecture. Their conceptual representation is presented in Figure 1.

The first one, introduced by Brooks [1] with the *subsumption architecture* is qualified as *reactive architecture* (Figure 1-a). This architecture is reactive to the environment stimuli but the global behavior of the architecture is unrepeatable. Hence, the control, and the evaluation of the system performances, in the context of complex missions is difficult. An arbiter [2] has been implemented to increase management capability of basic behaviors, in order to produce a coherent global behavior of the robot. However, global optimization is almost impossible.

To increase decision capabilities a second type of architecture called *deliberative architecture* is proposed (figure 1-b). This architecture presents generally three independent levels [3]. This decomposition of the architecture, and consequently of the abstraction level of information facilitates considerably the development of each part of the architecture. Nevertheless the data needs to go through all the layers of the architecture in order to reach the end of the decision/action chain (i.e. the actuators) which induces very low reactive capabilities.

(a) Reactive control architecture   (b) Deliberative control architecture
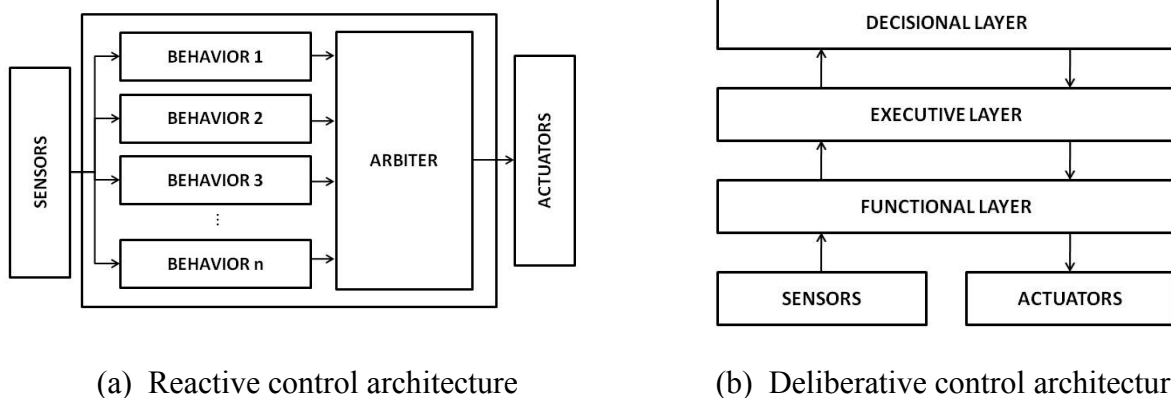
Fig. 1 – Historical control architecture

Those different types of architectures have opposite drawbacks. The principles exposed in the sequel propose to integrate in single control architecture the advantages of both previous types.

2.1.2   Hybrid control architecture

This type of control architecture is called *hybrid* or *mixed*. It is actually the most used type of architecture. Within this class, architectures may be very different.

For example, the LAAS architecture proposed by Alami in [4] is built as a deliberative architecture with three identified layers (figure 2-a). However the robot adaptation is made at

the decisional level, but can also be made at the functional level. Moreover the decisional level and the functional level are interconnected, while the executive control level manages the execution of the functional one. On the other hand, the CLARATy architecture [5] is a two layered architecture, where a reactive functional level and a deliberative level are interconnected (figure 2-b)

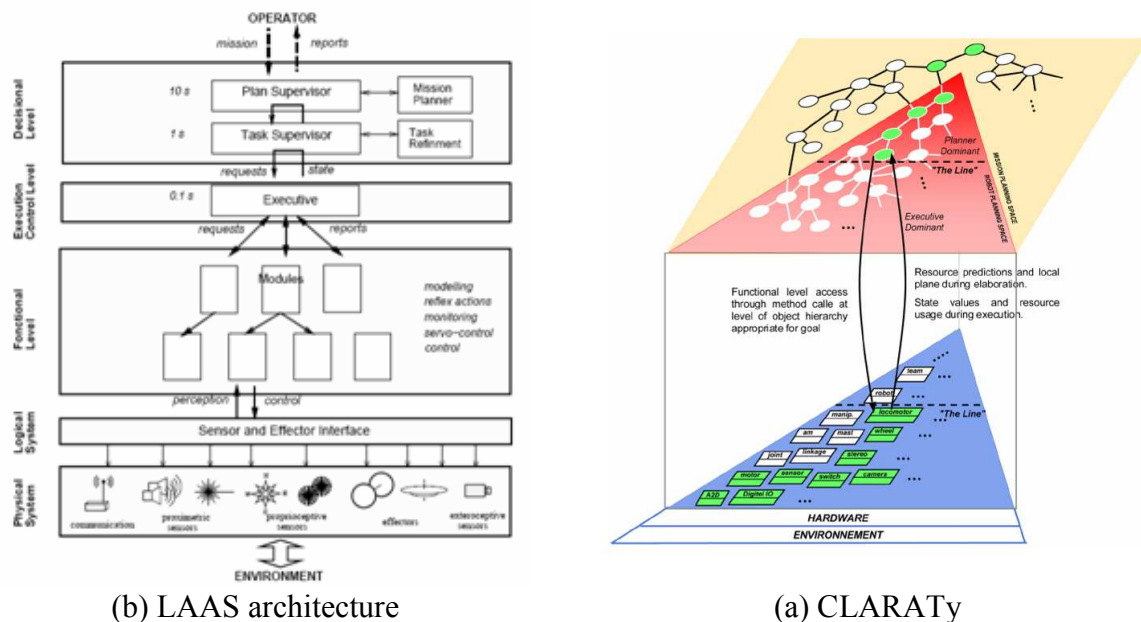(b) LAAS architecture   (a) CLARATy

FIG. 2 – LAAS and CLARATy hybrid control architecture

The mechanisms within these architectures are very specific to their architecture structure and their application domains. However, they lead to an important improvement in reactivity and decisional aspects.

2.2 Autonomy

Autonomy is one of the main objectives to achieve in mobile robotic. A lucid definition of *Autonomy* for a robot is difficult to find in the literature. In [6] Chopinaud does not succeed to provide a precise answer. But from a large study of the definitions used in robotics and multi-agent systems, the following one seems to be acceptable: "A system (or agent) is autonomous if, alone, it is able to define his decision". However this definition remains wide, and needs to be refined, from the point of view of applications focusing on autonomy evaluation.

The ALFUS group proposes in [7] a three-axis based evaluation: the operator independence, the mission complexity and the environmental difficulty. The results of all these researches remain either too general or too specific. In [8], Clough attempts to provide a generic tool to evaluate the autonomy of intelligent UAV (Unmanned Aerial Vehicle). He establishes an Autonomous Control Level (ACL) chart based on the degree of interaction between the robot and the human operator.

Despite the diversity of the literature references dealing with autonomy in robotics architectures, all the authors convey on the existence of several levels of autonomy. However, the definition of these levels is dependent on the point of view, the evaluation parameters or the application context. In [9], Goodrich *et al.* identify five different types of Human-Robot interactions: teleoperation, adjustable autonomy, mixed initiatives, advanced interfaces, autonomous robots. These types of interactions are similar with the ones which Dufour & al. define in [10], which also identify the *traded control* and the shared control where a part of the

mission or a part of the system is controlled by the human whereas the rest is controlled autonomously by the robot.

In these works, as proposed in [11], the autonomy levels of systems are related to the Human-Robot interactions degree. We follow this evident correlation and define (Figure 3) the autonomy levels we use in our context, based on the literature references and adapted for mobile robotic architectures.
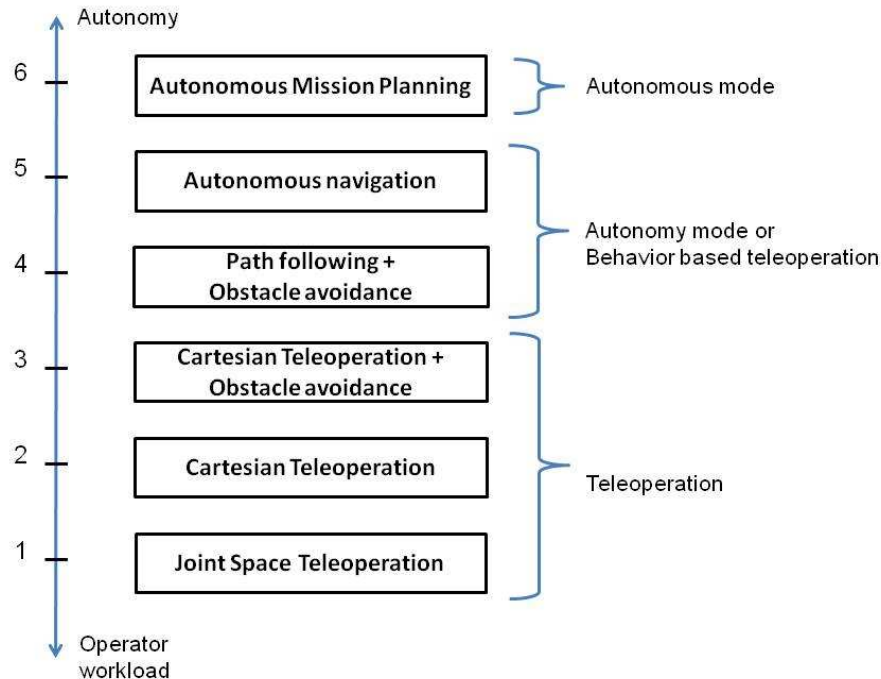

FIG. 3 – Autonomy mode classification

This classification is useful in our study related to adaptive autonomy. Indeed, it is well known that full autonomy for mobile robotic system in dynamic environment is a complex problem that is not yet solved. Then, occurrences of faults, as hardware failures, bad behaviors of control algorithms, real time failures or environment obstructions, prevent a full and complete autonomy.

This implies the dynamic management of different autonomy levels in single control architecture, and the control of commutation between these levels.

2.3   Adaptive autonomy

The first question to be solved is the difference between adjustable and adaptive autonomy. In [12] Goodrich distinguishes the adjustable autonomy, where the human can choose the functioning mode of the robot, and the adaptive autonomy where the functioning mode is chosen by the robot itself. Scerri et al. in [13] consider that adjustable autonomy refers to entities, able to dynamically define their own autonomy level. We do not make this difference, and the adaptive autonomy term in this paper means the modification of the functioning mode both by the human operator or by the robot, or even by a discussion between them.

Adaptive autonomy in robotic control architecture is an active field of research, but a complete and mature solution has not been yet proposed. Some partial solutions are proposed applying multi-agent architecture concepts to the robotic domain. In [14], Mourioux *et al.* present an architecture which allows for controlling a robot with various degrees of autonomy. But there is no dynamical management of these autonomy modes, depending for example on the context or the robot state. In [15], Stinckwich *et al.* propose the implementation of an anticipatory agent to allow for the adaptation of the architecture to environment changes or failures detection. But these works are still in progress and have not been implemented yet.

Furthermore, as for many multi-agent systems, the detection is done by a very specific agent dedicated to the targeted application. With the same idea, Dufour *et al.* in [16] propose an *attention manager agent* to supervise the execution of the other agents' execution. This solution is not described in details, particularly the supervision algorithms or the precise structural development.

Other studies, as the one of Mercier *et al.* in [17], are focusing on the authority sharing to avoid inconsistency due to conflict between the human and the robot decisions. But the first problem to be solved, for the adaptive autonomy implementation in robotic architecture, is the identification of the reasons why it is necessary to switch from an autonomy level to another. In this context, fault detection and diagnosis techniques are important to study.

2.4   Fault detection and Diagnosis

Because software or hardware faults cannot totally be avoided, it is essential that mobile robots are able to detect such faults. However fault detection, which uses observations and a model of the system's behavior to detect deviation between them, is not enough. The faulty hardware or software component must also be identified to solve the problem.

Many approaches for fault detection have been proposed and implemented. Model-based reasoning proposed by Reiter in [18], using a logic-based formulation of the system model and the observations, has been used for hardware fault detection by William *et al.* [19] and Friedrich *et al.* [20]. Steinbauer *et al.* in [21] have also chosen the model-based paradigm to detect and repair control software communication failure in runtime.

Some approaches take account of uncertainties of the robot and its environment to detect faults. For example, Verma *et al.* propose the use of particle filter techniques in [22] to estimate the most probable state of the robot and detect faults by comparison with the robot model. However, these detections do not provide any suggestion related to corrective actions. In [23], Brandstötter *et al.* propose an interesting approach where a probabilistic hybrid automaton models the nominal operational mode, 3 failure modes, and the probabilistic transitions between these modes, for an omnidirectional robot. But the detected fault remains limited (wheel problems) and the approach is not generic.

Moreover, in [24] Murphy *et al.* propose rule-based approaches to detect and recover failures in sensing. Kalman filters bank can also be used to detect specific failure mode, as proposed in [25], by Roumeliotis *et al*.

2.5   Conclusion

As far as we know, few works have been developed to **systematically** detect hardware and software faults and to propose relevant solutions during runtime. Moreover, these works focus on some specific application or specialized control structure, and do not propose modular and suitable control architecture to support their implementation. So they cannot be inserted into a **global and generic approach** dealing with fault detection monitoring and management. The proposed solutions are often locals, too specifics and difficult to use in others contexts.

This paper presents a generic and systematic solution to design efficient and robust control architecture for mobile robots with adaptive autonomy. This architecture proposes solutions to detect faults and solve the related problems in a systematic way. Our adaptive autonomy concept is based on detection mechanisms of robot inconsistencies, management of these errors detection, an adaptive strategy based on Human-Robot interaction, and a structural framework to switch between autonomy modes. Thus, when a fault is detected during the robot mission execution, either the robot can manage or it contacts the human operator who provides a solution adapting the autonomy of the robot. This paper deals with the first problematic of this adaptive architecture: the detection of faults.

3   TOWARDS AN EVALUATION OF ROBOT CONSISTENCY

During the execution of an autonomous robotic mission, many dysfunctions may occur:
- the physical parts of the robot (sensors, actuators or electronic devices) can have failures,
- internal algorithms dysfunctions or important computing load drift, can induce the non satisfaction of the real time constraints,
- some limitations or approximations in the models, and corresponding algorithms used to control or localize the robot can create erroneous internal data.

Some of these dysfunctions can induce the failure of some of the mission's objectives or the mission itself. Others only create local disturbance inducing loss of efficiency, while others can have minor impact and can be simply filtered.

These dysfunctions generally result in inconsistencies in the robot's state. We then can use inconsistency detection to identify dysfunctions and try to correct the problem. In some situations, the robot can manage itself or can call out to the Human supervisor which in turn can adjust the robot's autonomy to help the robot to manage his mission.

3.1 Inconsistencies: classification

Four main classes of state inconsistencies can be identified:
- Application data inconsistency: this class concerns instantaneous erroneous data produced by the robotic application.
- Global behavior inconsistency: the evolution of the robot over the time can produce inconsistency.
- Architectural inconsistency: this class concerns errors which can be produced by the architecture, as the non respect of real time constraints.
- External inconsistency: this class contains errors induced by the communication between the robot and the human.

The identification of inconsistencies can be done by different algorithms and techniques which are presented in Section 5.

3.2 Characterization of inconsistencies

To guide, assist and help the Human supervisor decision making, the robot sends information on the detected inconsistencies. The inconsistencies must then be characterized by a set of relevant data:
- Name: name of the detected inconsistency,
- Class: {hardware, software, models},
- Location: where, which sensor, actuator or algorithm,
- Criticality: level of criticality,
- Additional data: set of contextual relevant data including the ones that put in light the inconsistency.

The proposed approach of inconsistency detection must be supported by an efficient control architecture.

4   INTEGRATION IN AN HYBRID CONTROL ARCHITECTURE

In [26], ElJalaoui proposes an hybrid control architecture, used in the LIRMM laboratory, to control an autonomous underwater vehicle (AUV). This control architecture has been developed to respect modularity, reusability and upgradeability. It is based on two different levels: an executive and a decisional level. The executive level is supervised by a scheduler activating the appropriate robotic tasks in time. The decisional level only reacts on events coming from the executive level. The internal structure of the architecture is based on a generic module paradigm, and is detailed in the following sections.

4.1   The decisional level

The decisional level is composed with three layers (Figure 4): a global supervisor, a set of local supervisors (one for each autonomy level) and the scheduler.

The global supervisor controls the whole mission. From a given mission and events produced by the architecture, it defines an objective and sends it to a local supervisor. The local supervisor is in charge of the accomplishment of this objective. From the objective and from events produced by the architecture, the local supervisor gives to the scheduler a sub-objective to be executed.

The decisions taken by the supervisors and the scheduler are made regarding a database called *vocabulary*. The vocabulary contains the decomposition of the mission into objectives and of an objective into sub-objectives. It also contains relations of precedence between the executive tasks which allow the scheduler to administrate the executive level.
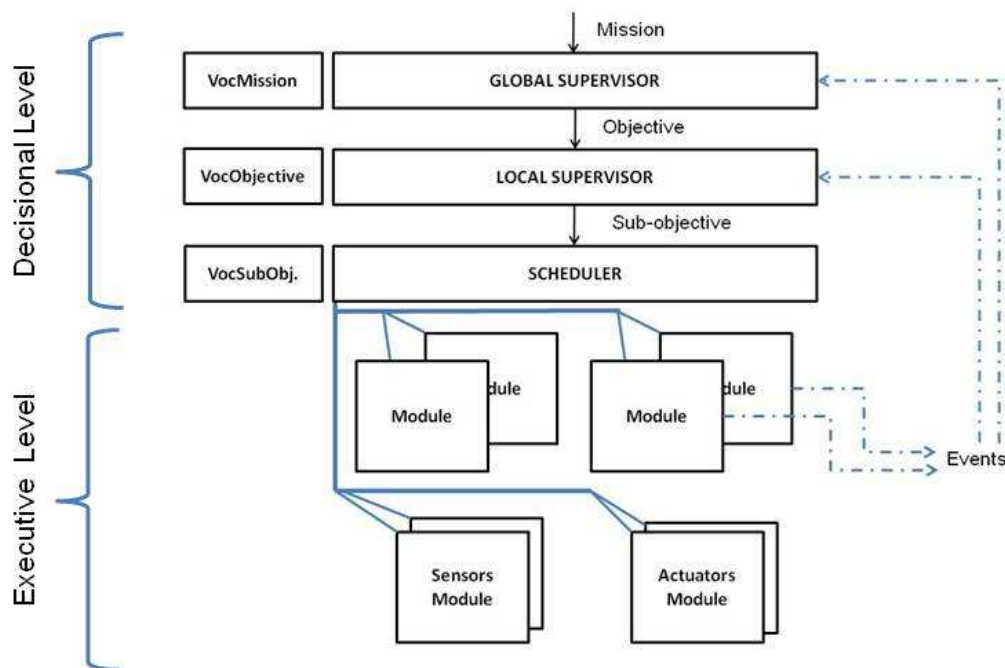

FIG. 4 – The LIRMM architecture

4.2   The executive level

The executive layer is based on the concept of module. Each module represents an independent task of the whole controlling software. To create a robotic task, modules have to be recursively activated over the time in a specific order. Then each sub-objective will use different modules with specific time constraints.

As an example a mobile system involved in a line following objective, only requires the recursive activation of the modules: sensors acquisition, line detection in the sensors

information, control law computation and actuators reference application. A more general mission related to mobile robotics requires the recruitment of a complete sensors suite, through the activation of the sensor modules, the execution of a path planning strategy and advanced control of reactive obstacle avoidance.

The modules of the executive level create events which are used by the supervisors to control the whole mission execution (Figure 4).

4.3   The infrastructure: a middleware

The infrastructure is a middleware developed as a library including the generic description of modules and the management of all the communications between them. A module is composed by a "System part" given by the middleware, and a 'User Part' in charge of the control architecture developer.

The "System part" manages the communication between modules in terms of data and event flow but also in term of process (activation, stop and configuration). The "User part" only contains the associated algorithm. This structure is very interesting in terms of modularity, reusability and upgradeability.

The communications between modules use intput/output communication ports. A generic module contains 6 types of port (Figure 5): Input data, Output data, Input event, Output event, Configuration Port and Request port (activation, stop, configuration, subscription and cancelation to data or events). Data and events are sent and received with the 4 first ports. The scheduler links producers and consumers with the configuration ports, and manages the execution of these tasks with the Request port. All communications between the modules respect the producer/consumer paradigm.
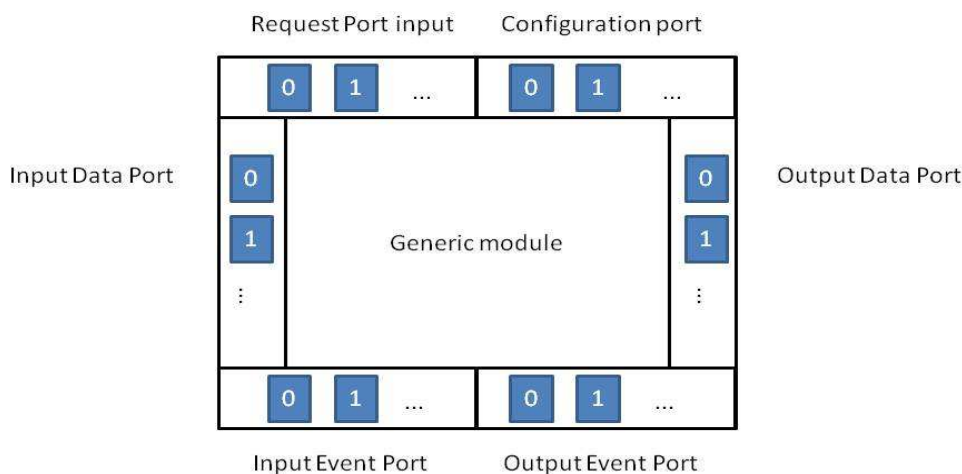

FIG. 5 – Generic representation of a module

The previous section presents the hybrid control architecture developed in LIRMM by ElJalaoui. In the context of autonomous robots control, we want to adapt this architecture to allow adaptive autonomy and Human-Robot interactions. The rest of the paper presents the developed principles used to integrate the detection and the management of inconsistencies into the LIRMM architecture.

5   OBSERVATION APPROACH

The concept of observation modules has been largely used in various scientific domains. For example in Microelectronic area observation modules are physically used on chips to capture some internal data for logical testing [27]. In organic computing systems observer / controller structure can be used to verify that an emergent behavior remains within predefined limits [28]. Model faults detection and isolation approaches can also be used to detect and isolate faults on real systems from the discrepancies between system outputs and model outputs [29].

In the LIRMM architecture, some architectural modules have been specialized into *Observation Modules* (OM) to detect the occurrence of the inconsistencies identified in Section 3. The systematical use of this type of module will create into the control architecture a new executive level dedicated to observation (Figure 6).

The basic observation functionalities are implemented into the specific Observation Modules which are connected to the robotic modules. The OM provides information about the consistency state of the system: the *Consistency Indicators* (CI). A specific observation module is the *Global Evaluation Module* (GEM). It evaluates the CI and returns relevant information to the higher levels (decisional level or human). The observation modules can be dynamically managed to respect real time constraints and to adapt the execution load on the system hardware.
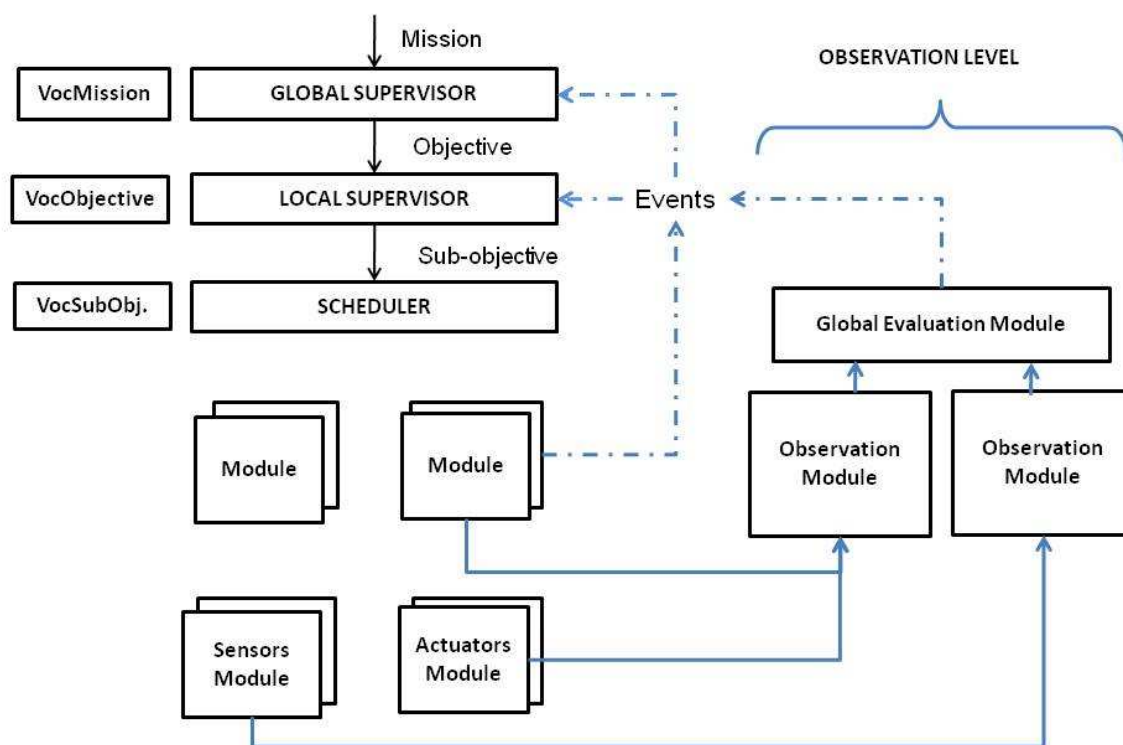

FIG. 6 – Integration of the observation level

The rest of this section presents in details the structural mechanisms for the implementation of this observation level. An applicative example is given in Section 6 to illustrate these principles.

5.1   Typology of the Observation Modules

The systematic use of Observation Modules implies the definition of design rules. This section defines the internal structure of the Observation Modules, their connection typology, and observation methods for the Consistency Indicators evaluation.

5.1.1 Internal structure

Basically, an observation module needs to consume relevant data to detect inconsistencies. It has, at least, one input port (Figure 7-a). Moreover it produces a data representing the evaluation of the Consistency Indicator (C.I.) of the observed functionality. It has one output port. To minimize the number of observation modules, it is possible to aggregate several basic observation modules sharing different input data (Figure 7-b). However, this aggregation process reduces the flexibility, the modularity and the readability of the proposed control architecture.
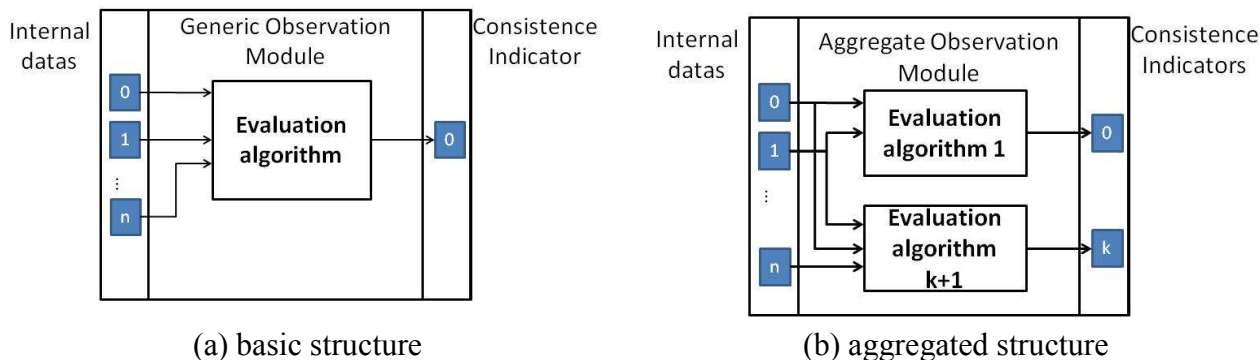

(a) basic structure   (b) aggregated structure

FIG. 7 – Internal structure of observation modules

5.1.2 Connection typology

The integration of the Observation Modules in the architecture can be classified according to their connection into the architecture. We define 3 types of connections, illustrated in Figure 8 on an architecture having typical control modules for the "path following" robotic task.

- Data Evaluation (D.E.): concerns the evaluation of data produced <u>or</u> consumed by only one single control module.
- Module Evaluation (M.E.): concerns the evaluation of data produced <u>and</u> consumed by <u>one</u> control module**.**
- Multiple Module Evaluation (M.M.E.): concerns the evaluation of data produced and/or consumed by <u>different</u> control modules.

The Consistency Indicators are produced by the Observation Modules according to the evaluation of the observed modules.
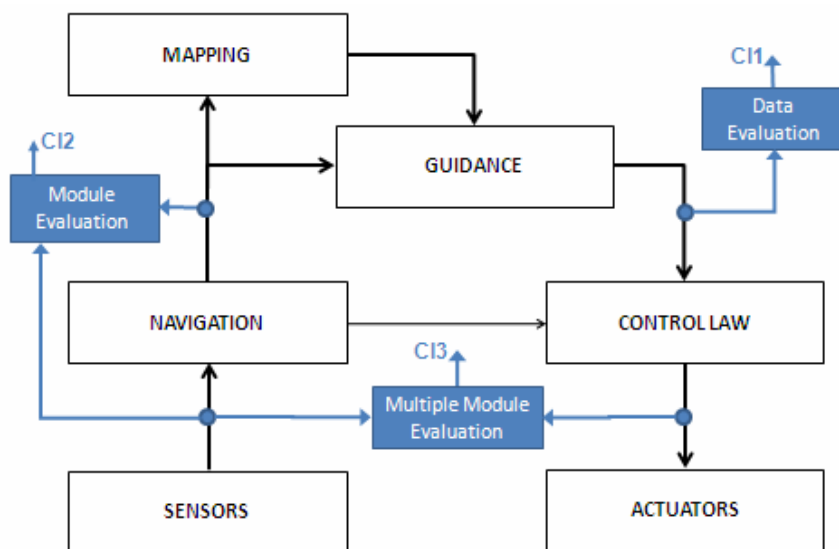

FIG. 8– Observation Module connection typology

5.1.3 Consistency Indicator evaluation

The evaluation of a Consistency Indicator is realized by integrating into the executing code of the Observation Module, a chosen algorithm corresponding to the detection of a given inconsistency. The proposed connection typology also defines a typology of the detection algorithms.

Data Evaluation (DE) modules contain typically threshold verification. They can be used to observe the respect of real time constraints, to evaluate the communication quality or to detect outranged control data. Threshold verification can be done in a static way, i.e. the data is compared with a threshold value, or in a dynamic way, i.e. the observed data is recorded and the observation is done over a given elapsed-time horizon. For example, in Figure 8, the DE module can detect if the reference produced by the Guidance module is out of the robot action capabilities.

Module Evaluation (ME) modules typically use model-based diagnosis algorithms applied to one specific elementary task. The ME and its observed module must produce the same data, within a given range. These data are then compared to detect inconsistencies. For example in Figure 8, the ME and the *Navigation* modules can use different localization methods. Navigation based on particle filter, produces a precise estimation of the system position. Since this estimation is based on a stochastic process, failure might happen. Then, the ME dedicated to the monitoring of the Navigation system, can embed a basic and imprecise, dead reckoning navigation strategy able to detect irregular jumps in the estimation provided by the Navigation system.

Multiple Module Evaluation (MME) modules can also use model-based diagnosis but in a most global point of view, comparing the behavioral evolution of the robot. For example on Figure 8 the MME could implement Multiple Models Kalman Filters (MMKF) [31] to monitor the physical part of the robot using *Actuators* input data and *Sensors* output data. The MMKF allows for detecting and identifying faults on wheeled mobile robot, as described in [25]. Another type of evaluation algorithm can be implemented to observe the group *Navigation–Guidance–Control* (NGC) and monitor the realization of the entire robotic task.

5.2 Global Evaluation Module

The Consistency Indicators produced by the Observation Modules are centralized in a dedicated Global Evaluation Module (GEM). The GEM aggregates the Consistency Indicators of the current active Observation Modules, and extract relevant information to be sent to the decisional level of the robot architecture. The aggregation of the Consistency Indicators is made using several criteria:

- the type of the inconsistency,
- the criticality of the inconsistency,
- the frequency of the inconsistency,
- the value of the Consistency Indicator.

In [23], Brandstötter *et al.* define three criticality levels: adjustable minor fault, important fault and critical fault. For an important fault, the robot can manage its mission with an adaptation. For a critical fault the robot is not able to manage its mission, and may need help from human operator.

Other important criteria for the evaluation of the criticality of inconsistencies are parameters related to the operator, like his actual role in the accomplishment of the mission. In context of multi-operator supervising a fleet of robots, the operator himself is source of many criteria as competence, experience and current workload.

Regarding the result of aggregation, the GEM can trigger different actions:

- *No action*. For example the first occurrence of a minor error.

- *A feedback control on the control sequence* (modules activated to control the robot). This concerns recursive minor behavioral inconsistencies with well-known solutions.
- *A feedback control on the evaluation sequence*. The evaluation sequence refers to the set of Observation Modules used to observe the control sequence. This can concern recursive minor real time inconsistencies.
- *A report to the operator* with the different Consistency Indicators involved in the fault and their corresponding information which characterize the consistency loss.

5.3   Real Time management / scheduling

5.3.1   Basic Scheduling

Figure 9 shows the normal activation (i.e. without observation) over the time of the control sequence of the "Path Following" robotic task. This sequence is recursively activated by the scheduler until an event changes the current sub-objective via a supervisor.
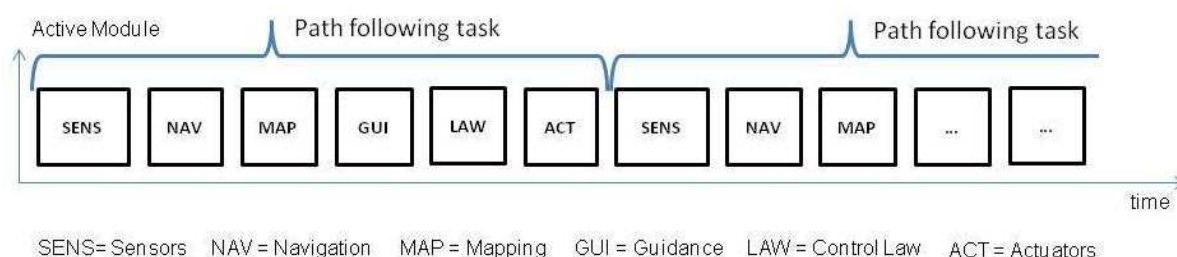

FIG. 9 – Modules scheduling for the "Path Following" robotic task

According to the observation approach proposed in this paper, many Observation Modules and a Global Evaluation Module are added within the previous modules activation sequence (Figure 10). Clearly, the position of the observation modules in the scheduling depends on the availability of their input data, and the Global Evaluation Module is executed at the end of the sequence.
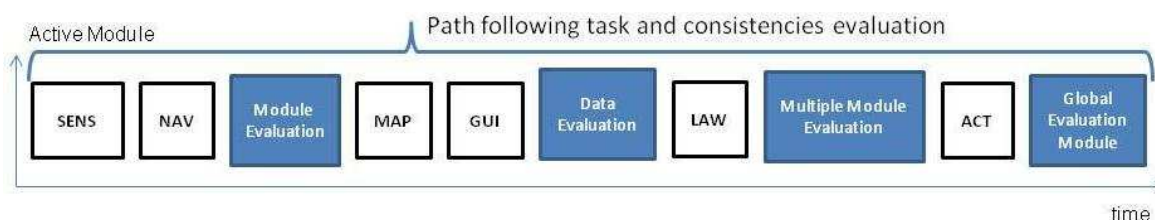

FIG. 10 – Modules scheduling integrating control and evaluation sequence

The comparison between Figure 9 and Figure 10 puts in light that the use of the proposed observation approach can add a large amount of new modules. This induces an important computational burden which has to be managed to satisfy the real time constraints.

5.3.2   Dynamical management

For a given robotic task and its corresponding control and evaluation sequence we propose two solutions to manage the activation of Observation Modules to respect real time constraints. These solutions are in accordance to the software mechanisms proposed by the LIRMM architecture. In the first proposition, the periodicity of activation of the Observation Modules is adjusted in order to respect real time constraint. Thus the OMs are not executed in each control sequence of the robotic task. The second proposition is to manage the number of active modules in the whole sequence: some OMs are suppressed of the evaluation sequence.

➢ Periodic activation adjustment

The first solution does not change the number of Observation Modules. It only adjusts their periodic activation according to the verification of real time constraints and to their criticality. The Global Evaluation Module dynamically modifies the periodic parameters of the OMs (Figure 11). The periodicity of the OMs activation is a parameter used by the scheduler to generate the modules scheduling.
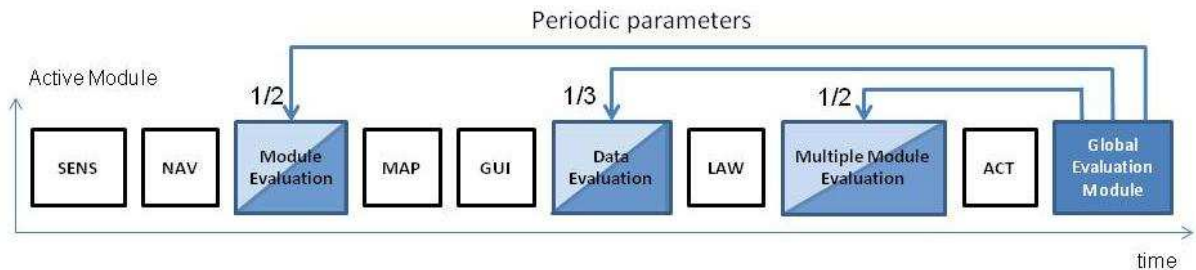

FIG. 11 – Periodic activation adjustment of the OMs

As an example the Module Evaluation and the Multiple Module Evaluation in Figure 11 are activated only once while the control sequence is activated twice, and the Data Evaluation module is activated each three periods of control sequence.

➢ Activation management

The second solution (Figure 12) is to "physically" eliminate (or add) some Observation Modules from the scheduling sequence, if necessary. If the Global Evaluation Module detects a problematic situation, it can generate a specific event to the local supervisor. Then this supervisor proposes a new sub-objective to the scheduler integrating less (or more) Observation Modules upon the same robotic task (and then within the same control sequence).
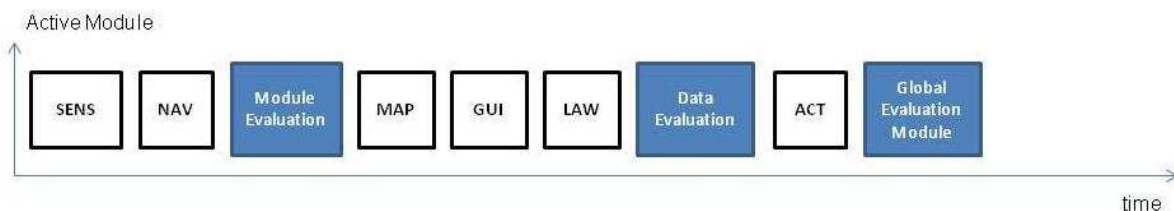

FIG. 12 – Activation management using a new sub-objective

6   EXPERIMENTAL PROPOSITION

The proposed approach and mechanisms are currently used to integrate adaptive autonomy for mobile robots in the hybrid control architecture based on the LIRMM architectural concepts.

The goal of the project is to develop a fleet of wheeled autonomous robots running into the laboratory to realize an office point to point delivery task from a sender to a receiver. First, the hardware characteristics of the robot are detailed. Then a global overview of the architecture under development is presented. Finally an example of mission is explained in detail to put in light the expected adjustable behavior.

6.1   Hardware characteristics

The project uses two robots Pioneer-3DX from MobileRobots inc©. This robot has a balance drive system (two-wheel differential with caster), reversible DC motors, motor-control and drive electronics, high-resolution motion encoders and battery power. It supports two sonar arrays, each with eight transducers that provide proximity information over 360 degrees. Bumpers provide contact detection.

A microcontroller manages sensors and actuators. In the present project an embedded PC is connected to the robot microcontroller for low-level robot control. The robot-control software proposed by MobileRobots inc$^{©}$ is not used. The connection between the on-board PC, where the proposed control architecture is running, and the microcontroller is implemented using a client-server communication.

The embedded PC Dual-Core works at 2 GHz and supports Wifi or Bluetooth communication facilities. The real-time control architecture implementation uses Linux RTAI operating system. The control software architecture development is realized in C language and uses the development facilities available from the LIRMM architecture middleware.

6.2 Architecture overview

A simplified description of the global architecture organization planned for the project is proposed in Figure 13. This architecture is currently under development. The different mechanisms described in this paper (fault detection and dynamic management) have been validated on simple examples. Just the basic control architecture has been implemented on the on-board PC. So unfortunately the Human-Robot interaction is not yet available.
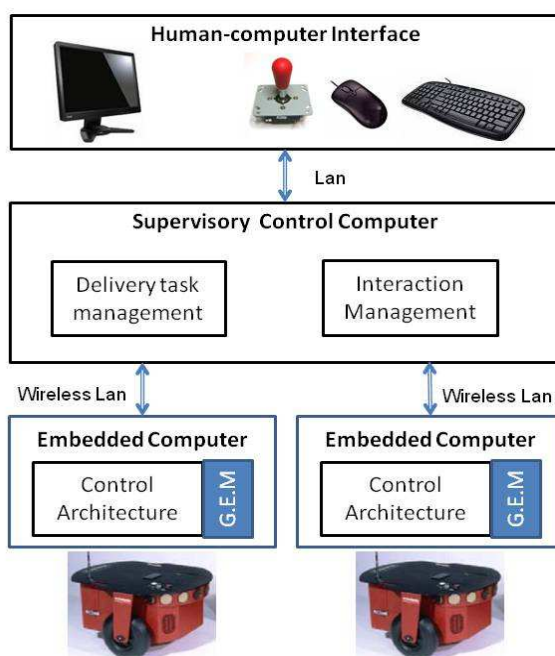

FIG. 13 – Simplified architecture organization

For the planned architecture three main parts can be distinguished:
- The lower part concerns the robots and their dedicated control architecture (including OM and GEM) implemented on embedded PC.
- The middle part corresponds to the supervisory part of the control. It concerns the global management of the office delivery task, and the Human-Robot interaction including inconsistency and generic data management.
- The higher part concerns the Human-Robot interface where visualization facilities and joystick control are available for teleoperation.

The delivery task management defines the mission attribution in function of the current missions, robots workload and their position. These criteria can be enriched in future development.

In order to be able to address adaptive autonomy the following autonomy modes will be implemented on the robots:

- Autonomous: dynamic mission planning and path following with obstacle avoidance.
- Path following with obstacle avoidance.
- Cartesian teleoperation with local autonomous obstacle avoidance
- Joint space teleoperation

In autonomous mode, from a mission given by a user via the delivery task manager the autonomous robot has to plan its mission, find the sender and then deliver the packet to the receiver in a full autonomous way. In path following mode, the robot follows a path defined by the operator onto the laboratory map shared with the robot. In Cartesian teleoperation the operator sends references to the robot, in terms of desired heading and speed, while the robot autonomously modifies the received references in order to avoid obstacles. In joint space teleoperation, the operator acts directly on the speed of each wheel of the robot without any closed loop controller executed by the architecture.

6.3   Example of an experimental mission

In the following example, the delivery office mission is made by two autonomous Pioneer-3DX robots carrying the same architecture, without collaborative mode. This mission assumes that all the location of the laboratory members is known, and that a distant operator can interact with the robots in case of inconsistency detection.

All the laboratory member(s) can post requests to send an object to some other laboratory member(s). When a request is detected, one of the robots comes to deliver the object according to its workload and its current position.

Figure 14 describes a realistic scenario which can be imagined within a mission. This puts in light how the detection of inconsistency, the autonomy adaptability and the Human-Robot interactions are connected.
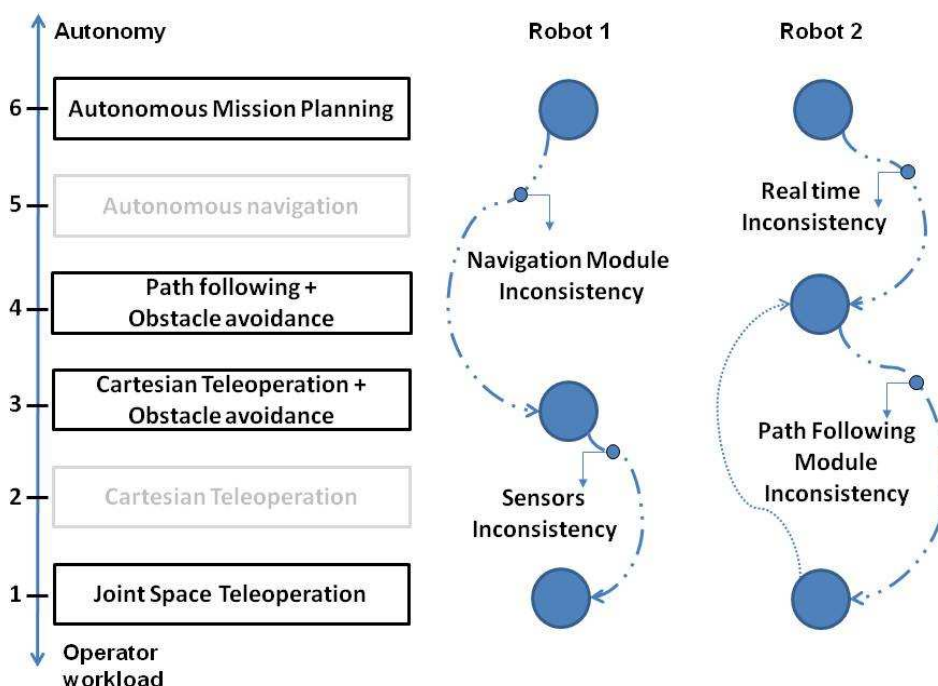
Fig. 14 – Scenario example: Autonomy mode activation

At the beginning this scenario considers that both robots receive their own delivery mission (autonomy level: 6, cf. Fig. 14), posted by 2 senders. Robot 1 plans autonomously the path from its current position to Sender 1 and reaches autonomously his location. Concurrently, Robot 2 returns to the operator the following information: detection of *real-time inconsistencies* and *failure of the path planning module* in the autonomous generation of the

path. Then, to overcome the detected problem, the operator defines a path for Robot 2 from its current position to Sender 2. The autonomy level of Robot 2 is decreased to 4.

After the reception of its packet, Robot 1 computes a path to reach the location of Receiver 1. On its way, an *inconsistency in the Navigation process* has been detected (one could imagine that the Navigation module, embedding a localization algorithm based on particle filter, resulted in an unreachable position estimation - in comparison with the result of a classic and imprecise dead reckoning algorithm run by the related Observation Module.) This fact is confirmed by the localization data sent to the Operator. To handle this situation the operator decides to drive Robot 1 to Receiver 1 using Cartesian Teleoperation (including autonomous obstacle avoidance, autonomy level: 3) and visual feedback. While being teleoperated, Robot 1 detects a *sensor dysfunction* on one of its ultrasonic proximeters (one can also imagine that this was the cause of the jump in the erroneous result of the navigation algorithm). Consequently, the Operator disables the automatic obstacle avoidance, decreasing the autonomy level to 1, and drives Robot 1 to Sender 1, or removes Robot 1 from the set of available resources – in this situation the mission of Sender 1 is transferred to Robot 2 agenda.

While navigating into a corridor, Robot 2 emits an inconsistency alarm to the Operator, triggered by an MME module which detected a *problem in the realization of the path following* objective of Robot 2: *uncharted obstacles* appeared and the nominal path is no more achievable. Since the level of autonomy of Robot 2 is 4, the autonomous path re-planning capability is disabled – a full autonomy (level 6) could handle this situation. To understand the situation, the operator can use other sensors, like camera, to find the cause of this problem. He can for example observe that the weekly meeting is over, and a lot of laboratory members are walking through the corridor. The operator then reduces the autonomy level of Robot 2 to 1 and carefully drives the robot out of the corridor. Then Robot 2 can return in the path following mode (level 4).

The above scenario illustrates how the robot control architecture can ask the operator to modify its current autonomy level. But furthermore the embedded architecture can also propose to terminate the teleoperation if the consistency evaluation process is able to detect that the inconsistency disappears.

7   CONCLUSION

This paper presents a generic and systematic observation approach included into a hybrid control architecture with adaptive autonomy for mobile robots. This approach adds an observation level to the architecture to detect and evaluate robot's state inconsistencies. This level is composed of specific Observation Modules (OM) and a Global Evaluation Module (GEM). These modules are a specialization of the conceptual module proposed in [9]. The Observation Modules are in charge of detecting the occurrence of inconsistencies. Inconsistencies have been distinguished (data, behavior, architectural or external) and characterized (relevant information). The OMs' role is to monitor and analyze internal data of the control sequence, in order to evaluate the Consistency Level of the robot. The analysis can be done using dedicated simple or sophisticated evaluation algorithms. Then the OMs send the Consistency Indicators to the Global Evaluation Module. The GEM is in charge to aggregate these Consistency Indicators and to manage the reaction strategies for the detected inconsistencies. It also identifies the most relevant information which must be sent to the Human supervisor. The supervisor can then adjust the robot autonomy according to the encountered problems.

The proposed approach increases significantly the number of modules embedded into the scheduling sequence. Then, to optimize the observation process according to the real time constraints, two dynamical management techniques are proposed: the periodic activation

adjustment of the Observation Modules using GEM control and the reconfiguration of the evaluation sequence.

These mechanisms are implemented in an hybrid control architecture developed to manage several autonomous mobile robots realizing an office delivery task. Two robots with 4 autonomy levels are available. To increase the global system efficiency, a distant Human supervisor can adjust the robots' autonomy in case of inconsistencies detection. Two different scenarios are described to illustrate the possible working of the control architecture with the proposed inconsistencies evaluation mechanisms.

To complete the works presented in this paper, the proposed dynamical management techniques has to be implemented. Then a rigorous evaluation of these techniques must be done to evaluate their relevance according to the impact on real time constraints. Furthermore, scheduling modifications and Human interventions are not the only possible reaction strategies to detected problems. Others strategies can be used to modify the robot control and to strongly enhance its autonomous behavior robustness.

Finally the definition of a generic framework would be helpful to guide the developer reasoning for the identification and the characterization of inconsistencies. This framework would facilitate the identification of pertinent architecture variables, connection typology and evaluation algorithms to easily design the Observation Level.

References

[1] R. A., Brooks "*A Robust Layered Control System for a Mobile Robot*", IEEE Journal of Robotics and Automation, 2 (1), pp 14-23, 1986.

[2] J. K Rozenblatt., *"DAMN: "A Distributed Architecture for Mobile Navigation""*, Ph. D thesis, 1997.

[3] R. P. Bonasso, R.J. Firby, E. Gat, D Kortenkamp., D.P. Miller, and M. G. Slack, "*Experiences with an Architecture for Intelligent, Reactive Agents*", Journal of Experimental and Theoretical Artificial Intelligence 9(2): 237-256, 1997.

[4] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "*An architecture for Autonomy*", International Journal of Robotics Research, 1998.

[5] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "*CLARATy: Coupled layer architecture for robotic autonomy*", Technical report, Jet Propulsion Laboratory, 2000.

[6] C. Chopinaud, *"Contrôle de l'émergence de comportements dans les systèmes d'agents cognitifs autonomes"*, Thèse de doctorat de l'université Pierre et Marie Curie, 2007.

[7] H-M, Huang, K. Pavek, B. Novak, J. Albus, and E. Messina, "*A Framework For Autonomy Levels For Unmanned Systems (ALFUS),*" Proceedings of the AUVSI's Unmanned Systems North America 2005, June 2005, Baltimore, Maryland.

[8] B. T. Clough, *"Metrics, Schmetrics! How The Heck Do You Determine a UAV's Autonomy Anyway?"*, Performance Metrics for Intelligent Systems Workshop, Gaithersburg, MA, USA, 2002.

[9] M. Goodrich, D. Olsen, J. Crandall, and T. Palmer, *"Experiments in adjustable autonomy"*, In Proceedings of IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Intelligent Agents, 2001.

[10] D. Dufourd, and A. Dalgalarrondo, *"Integrating human / robot interaction into robot control architectures for defense applications"*, CAR 2006.

[11] T. B. Sheridan, and R. Parasuraman, *"Human-Automation Interaction"*, in book Reviews of Human Factors and Ergonomics, Volume 1, Edited by Raymond S. Nickerson.

[12] M. Goodrich, T. W. McLain, J. D. Anderson, J. Sun, J. W. Crandall, *"Managing autonomy in robot teams: observations from four experiments"*, HRI 07, pp 25-32, 2007.

[13] P. Scerri, D. V. Pynadath and M. Tambe, *"Towards Adjustable Autonomy for the Real World"*, Journal of Artificial Intelligence Research - JAIR, vol 17, pp 171-228, 2002

[14] G. Mourioux, C. Novales and G. Poisson, "*Control robot by a generic control architecture*", proceedings of IROS, 2007.

[15] S. Stinckwich, and N. Bouraqadi, "*Towards an Adaptive Robot Control Architecture*", CAR 2007, AROUND Project.

[16] D. Dufourd, and A. Dalgalarrondo, *"Integrating human / robot interaction into robot control architectures for defense applications"*, CAR 2006.

[17] S. Mercier, F. Dehais, C. Lesire, and C. Tessier, "*Resources as basic concepts for authority sharing*", HUMOUS, 2008.

[18] R. Reiter, "*A theory of diagnosis from first principle*", Artificial Intelligence, 32(1), pp. 57-95, 1987.

[19] B. C. William, P. Nayak, and N. Muscetolla, *"Remote agent: To bodly go where no AI agent has gone before"*, Artificial Intelligence, 103(1-2), pp. 5-48, 1998.

[20] G. Friedrich, M. Stumptner, F. Wotawa, *"Model-based dignosis of harware designs"*, Artificial Intelligence, 111(2), pp. 3-39, 1999.

[21] G. Steinbauer, M. Mörth, and F. Wotawa, *"Real time diagnosis and repair of faults of robot control software"*, Robocup Intenational Symposium, pp. 13-23, 2005.

[22] V. Verma, G. Gordon, R. Simmons, and S. Thrun, *"Real-time fault diagnosis"*, IEEE Robotics and Automation Magazine, 11(2), pp. 56-66, 2004.

[23] M. Brandstötter, M. W. Hofbaur, G. Steinbauer, and F. Wotawa, *"Model-Based Fault Diagnosis and Reconfiguration of robot drives"*, IROS, in proc. pp. 1203-1209, 2007.

[24] R. R. Murphy, and D. Hershberger, *"Classifying and recovering from sensing failure in autonomous mobile robot"*, AAAI/IAAI, vol. 2, pp. 922-929, 1996.

[25] S. I. Roumeliotis, G. S. Sukhatime, and G. A. Bekey, "*Fault Detection and Identification in a Mobile Robot using Multiple-Model Estimation*", Proc. of 1998 IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 2223-2228, 1998.

[26] A. ElJalaoui, "*Gestion Contextuelle de tâches pour le Contrôle d'in véhicule sous-marin Autonome*", Thèse de doctorat de l'université Montpellier 2, 2007.

[27] W. Laung-Terng, W. Cheng-Wen, and W. Xiaoqing, "*VLSI Test Principles and Architectures: Design for Testability*", Elsevier, 2006.

[28] T. Schöler, and C. Müller-Schloer, "*An Observer/Controller Architecture for Adaptive Reconfigurable Stacks*", pp139-153, ARCS 2005.

[29] L.F. Mendonça, J.M. Sousa, and J.M. Sá da Costa, "*An architecture for fault detection and isolation based on fuzzy methods*", Expert Systems with Applications: An International Journal 36(2): 1092-1104, 2009.

[30] C. Plagemann, C. Stachniss, and W. Burgard, "*Efficient Failure Detection for Mobile Robots Using Mixed Abstraction Particle Filters*", EUROS, Palermo, Italy, 2006.

[31] R. Dearden, F. Hutter, R. Simmons, S. Thrun ,V. Verma, and T. Willeke.: "*Real-time fault detection and situational awareness for rovers: Report on the Mars Technology Program Task*", Proc. of IEEE Aerospace Conf., vol. 2, pp. 826–840. IEEE Computer Society Press, Los Alamitos, 2004.