2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

HiDDeN: Cooperative Plan Execution and Repair for Heterogeneous Robots in Dynamic Environments

Thibault Gateau
ISAE / CAS
31055 Toulouse, France
Thibault.Gateau@isae.fr

Charles Lesire
ONERA- The French
Aerospace Lab
31055 Toulouse, France
Charles.Lesire@onera.fr

Magali Barbier
ONERA- The French
Aerospace Lab
31055 Toulouse, France
Magali.Barbier@onera.fr

***Abstract*— This paper presents HiDDeN, a high-level distributed architecture for multi-robot cooperation. HiDDeN aims at controlling a team of heterogeneous robots in environments with uncertain communications. It relies on a mission plan defined as an instantiated HTN, i.e. a hierarchical decomposition of robots' tasks. This hierarchical structure also benefits to plan repair operations in case of failure detections. This repair is made as local as possible, in order to avoid unnecessary communications between robots.**

I. INTRODUCTION

Robots have already reached a high level of autonomy in term of decision making. Most of the time, this autonomy relies on a well-defined embedded software architecture [1], [2], [3], [4], [5], [6]. Most of these architectures are currently in use for complex real-world situations where autonomous robots must manage the execution of their own tasks.

The complexity of missions increases with the development of robots and of their abilities. The progress of the work in this field has led researchers wonder for many years to what extent autonomous heterogeneous robots with complementary functions would be able to cooperate as a team. We can already assume that robots, and therefore the team, will be deployed incrementally, due to the cost and the time required for setting up such technologies. For instance, in spatial exploration, robots (rovers, satellites, etc.) are deployed one after the other, and missions can last a couple of decades. Robots will then be intrinsically heterogeneous, as they will be developed gradually, and probably by different manufacturers. Hence, their embedded architecture may be different. Therefore, it would be worthwhile to reuse all the robots' skills, often developed in a mono-robot concern, to allow a team of robots to achieve a specific mission cooperatively, without re-designing all their control architecture.

Even assuming that this interoperability issue is solved, the dynamic aspect of the environment remains a key problem for a team of robots. The initial plan can take into account some constraints but not all the uncertainties on actions' achievements and environment changes. When executing the mission in a real context, failures *will* occur in the plan, as the robots will have to face real unexpected events. At the decision level, for failures concerning more than one robot, a plan repair procedure has to be defined from the point of view of the team.

In this paper, we propose a distributed multi-robot architecture for team cooperation that deals with two major issues:

- cooperation between heterogeneous robots while retaining their own embedded control architecture;
- minimal communication needs by:
  - distributing cleverly the mission plan to each robot;
  - applying a distributed plan repair strategy.

This paper first presents some of the existing multi-robot architectures, focusing on how they handle communications. Afterwards, we describe our HiDDeN architecture. We finally present a successful experiment campaign.

II. STATE OF THE ART

The BErkeley AeRobot (BEAR) project[1] studies multi-agent probabilistic pursuit-evasion games with heterogeneous robots. They developed a distributed, hybrid and hierarchical architecture system in order to manage partial knowledge of states among the team [7]. They take into account a dynamic environment, heterogeneous agents, fault on robots, and sensor imperfections. Contrary to the *sense-plan-act* architectures, robot's dynamics is taken into account at a higher level of the decision process, and they take a particular care to limit communications to the minimal needs. However, in spite of the modular aspect of this architecture, connections between modules are still numerous, which leads to a complex adaptation process to re-use existing mono-agent architectures.

The ALLIANCE architecture [8] is a behavior-based approach to cooperation and allows the robot team members *to respond robustly, reliably, flexibly and coherently to unexpected environmental changes*. They demonstrate successfully the feasibility of their architecture in an implementation example. In our work, we lean upon some of their ideas, even if we are not working with motivational behavior. In the same way, Chaimowicz [9] proposes an architecture system for tightly coupled multi-robot cooperation. The robot team is organized in a flexible leader-follower structure in which the local robot architecture is independent from the robot control manager. However, permanent communication is required.

Tambe [10] points out that *the more the team is flexible and robust, the higher communication load is required*. He

[1] http://robotics.eecs.berkeley.edu/bear

thus emphasizes the importance of managing the communication in team mission execution, as it represents the most critical resource and may lead to a mission failure. Several approaches study the management of the team mission execution with communication constraints. STEAM (a Shell of TEAMwork) [10] proposes a general model of teamwork where the mission is built around the teamwork core, and not conversely. STEAM is based on the Joint Intention Theory. Similar work is done in [11] and [5], where the architecture is based on the BDI framework. This framework may bring consistency issues when communications are interrupted. For the moment, it seems difficult to overcome this problem using this framework.

Based on Contract Net Protocols [12], some authors have focused their work on allocating the mission tasks among agents [13], [14]. These studies are explicitly avoiding as much as possible the loss of communication between agents by allocating some of them to the maintenance of local communication links. This issue is then divided into local communication constraints, leading to an even more critical situation when data transfer cannot be ensured during execution. Therefore, this method cannot be applied in contexts where communication is uncertain.

Temporary lack of communication can be experienced during the mission execution. Joyeux [15] proposes a plan execution that allows situations where communication is not permanent. However, like in [16], the plan is distributed to the agents, but there is no local or global repair. Sotzing [17] takes communication constraints in consideration in multi-AUV operations. Mission execution and multi-robot coordination are supported by BIIMAPS (BlackBoard Integrated Implicit Multi-Agent Planning Strategy). All robots have a complete copy of the BIIMAPS plan, so robot actions can be predicted when there is no communication. When communication is anew available, robots refresh the knowledge coming from other team members.

Our contribution allows a robot to react to a disturbing event and makes plan repair as local as possible, based on a hierarchical organization of a global mission plan. We assume that communication is unpredictable. Contrary to Legras and Tessier [18] who define a team depending on communication contact, we define a team by robots' participation into the achievement of a task. We also want to keep an explicit view of the sub-teams' plans. Finally, we want to reuse local architectures that already exist and are mature for local control of a robot.

III. THE HIDDEN ARCHITECTURE

In this paper, we propose HiDDeN, a High level Distributed DecisioN layer for multi-robot mission monitoring. Each robot has a HiDDeN instance, the *local supervisor*, which is interfaced with the existing robot control architecture (Fig. 1). This abstract layer chooses which action must be executed by each autonomous robot of the team, while controling coordination between teammates. Figure 2 shows a global plan of a HiDDeN *local supervisor*, from a robot's point of view. *Local supervisors* are decomposed in a modular structure, according to most of the architectures in the state of the art. If interfaces are well defined, modularity brings good genericity, possibility of unit testing and means to implement them separately. These modules are described in the following.

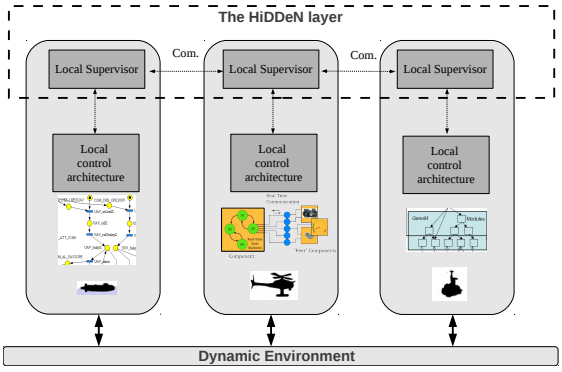


Fig. 1: The HiDDeN layer for monitoring a team of robots.

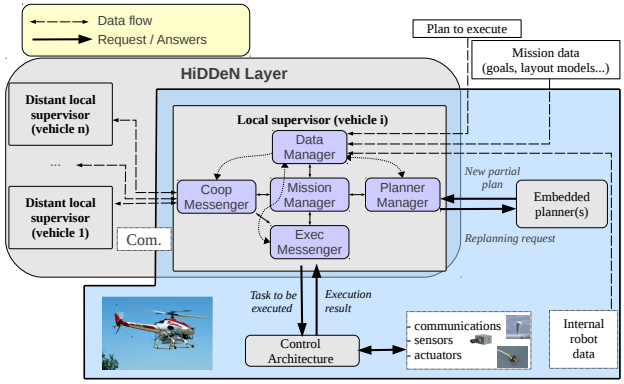


Fig. 2: Overview of a local supervisor.

A. `DataManager`: *providing mission-related information*

The **`DataManager`** is responsible of updating and providing information about the mission. It relies on Koper [19], an instance of an ontology that contains information about the robots' states, the current plan, the robots' actions, the mission goal, etc. It distinguishes local data (i.e. managed by the robot the Koper instance is embedded on), that are directly get/set from the robot control architecture, and remote data (i.e. managed by other robots) that is updated when a communication is explicitly made with teammates.

B. `ExecMessenger`: *interfacing with control architectures*

The **`ExecMessenger`** module represents the interface of the *local supervisor* with the control architecture of the robot. In general, autonomous robots own a specific communication protocol allowing interaction with a human operator, or at least, an internal supervision system that indicates which task has to be achieved. The current task selected by the **`MissionManager`** is then translated by **`ExecMessenger`** into a form understandable by the local architecture. Execution results are then translated from the local architecture to the **`MissionManager`**. This interaction

is also supported by Koper [19], where the services provided by the robots are described. This description contains the name of the services, their parameters, their pre/post conditions, and the protocol used for calling them on the control architecture.

C. `MissionManager`: managing plan execution

The **`MissionManager`** module is the core of the *local supervisor*. It manages the plan execution, detects failures, and triggers synchronization tasks between robots and the repair process. Its behavior is detailed in section IV-C.

D. `PlannerManager`: hierarchical plan repair

The **`PlannerManager`** takes care of the interaction between the local supervisor and the embedded planners. These planners allow a repair or replan of the mission plan according to the encountered failures and according to the current situation. Its behavior is detailed in section IV-E.

E. `CoopMessenger`: communication and synchronization

The **`CoopMessenger`** deals with synchronization exchanges and is compulsory for team cohesion. It allows *local supervisors* to interact, using communication means provided by the autonomous robots. Its behavior is detailed in section IV-F.

IV. PLAN EXECUTION AND REPAIR

A. Plan Representation

The HTN (Hierarchical Task Network) formalism [20] is used to model the mission plan. An HTN is a hierarchical set of abstract and elementary tasks. One or more methods are assigned to an abstract task and describe the way to achieve it, using other abstract or elementary tasks. The selection of a method is constrained by the fulfillment of preconditions. The advantages of HTNs are their flexibility, their hierarchical structure and their convenient modeling of human knowledge. Besides that, HTNs are classically used to represent planning problems and some multi-agent planners have been specifically developed to deal with this formalism [21]. HiDDeN directly uses an *instantiated HTN* of the mission plan (noted iHTN in the following) as the core model of plan execution monitoring.

Definition 1. *An iHTN $\mathcal{H}$ is a tuple $(E, V, Pre, Post, Te, Ta, \Re, M, t_r)$:*

- *E is the set of labels of tasks and methods;*
- *V is the set of instantiated variables;*
- *Pre is the set of preconditions;*
- *$Post$ is the set of postconditions;*
- *$Te \subset 2^{Pre} \times 2^{V} \times 2^{Post}$ is the set of elementary tasks. An elementary task t_e is a triplet $(Pre(t_e), V(t_e), Post(t_e))$ with $Pre(t_e)$ a list of preconditions, $V(t_e)$ a list of parameters, and $Post(t_e)$ a list of post-conditions;*
- *$Ta \subset 2^{V} \times 2^{M} \times 2^{Post}$ is the set of abstract tasks. An abstract task t_a is a triplet $(V(t_a), M(t_a), Post(t_a))$ with $V(t_a)$ a list of parameters, $M(t_a)$ a list of methods, $M(t_a) \neq \emptyset$ (to which preconditions are associated), and $Post(t_a)$ a list of post-conditions;*
- *$\Re$ is a set of partial ordered relations applied to the set of tasks $(Ta \cup Te)$. We consider that $rel \in \Re$ is either sequential (noted $rel = \prec$) or non-ordered (noted $rel = \sim$);*
- *$M \subset Ta \times 2^{Pre} \times 2^{(Ta \cup Te)} \times \Re$ is a set of methods. A method m is a quadruplet $(t_m, Pre(m), st, rel)$ with t_m the abstract task to which the method is applied, $Pre(m)$ a list of preconditions, st a set of tasks, rel a sequential or a non-ordered relation between the st elements. rel provides a mean to define an execution order for tasks of the st set;*
- *t_r is the root abstract task that must be executed; this is the highest level of abstraction in the $\mathcal{H}$ tree.*

Note that this definition is similar to the general HTN definition: an iHTN is a subset of an HTN where the planner previously decides the methods and the variables to use. With deterministic planners (as the one we actually use), the plan contains one method per abstract task.

B. Plan Distribution

The team mission plan is supposed to be computed offline, prior to mission execution, for instance using a classical HTN planner [20]. As the HiDDeN supervisors are distributed among the team robots, this plan has to be distributed to robots. We propose a distribution process that will help minimize communication needs by:

- removing unneeded tasks from the plan, leading to a local plan adapted to the robot. This cleaning process helps maintaining plan consistency during plan repair: modifying the plan of a subteam has no impact on the plan of robots not involved in the repaired task.
- adding necessary communication tasks that are implicit in the mission plan, determined by dependencies between tasks achieved by several robots.

More precisely, the global plan $\mathcal{T}$ is distributed to the team of robots, providing each one with a local plan $\mathcal{H}$. For a robot r, this local plan is computed using an algorithm summerized in the following process:

1) when a method is unordered, the sub-tasks in which r is not involved are removed;
2) when a method is ordered, the sub-tasks in which r is not involved that do not immediately precede a task in which r is involved are removed[2]; other sub-tasks not involving r are replaced by synchronization tasks, i.e. robot r has to wait a communication from the distant robots which are responsible of these sub-tasks.

The distribution algorithm scans the iHTN in a bottom-up manner (from elementary tasks to the root task), applying previously described operations, and removing methods whose all sub-tasks have been removed.

The resulting robot local plan is then a reduced iHTN that contains either original tasks that directly concern the

[2]These tasks have no direct influence on a task of r.

robot, or generated synchronization tasks useful to maintain the precedence dependency between distributed tasks. Note that original tasks may also contain communication tasks that have been planned, contrary to synchronization tasks that are not present in the original team plan.

C. Plan Execution

Plan execution is carried out by the **`MissionManager`**. It executes the robot local plan by scanning it in a depth first (i.e. descending to elementary tasks) and left first (i.e. enforcing the ordered relations between tasks) manner (Fig. 3).

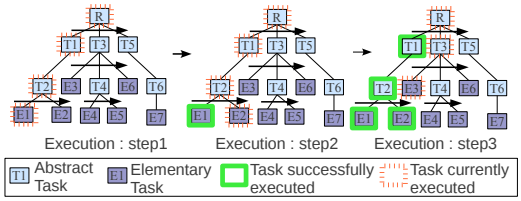


Fig. 3: Execution of an iHTN: HiDDeN processes root task R, descends to task $E1$. When $E1$ is successfully achieved, the following task is $E2$, then $E3$, and so on as long as no fault is detected.

When an elementary task (a leaf) is reached, the corresponding action is executed. This execution is devoted to the **`ExecMessenger`** if the task is a robot task (resulting in a request sent to the control architecture) or to the **`CoopMessenger`** if the task is a synchronization task.

D. Failure Detection

During the execution of the mission plan in a real context, faults or disturbances may occur. We introduce several means to detect such failures in HiDDeN. This failures are classified as follows:

- **invalid preconditions:** before executing a task, the **`MissionManager`** checks that the task preconditions are fulfilled, i.e. that the action can be executed in the current state; otherwise, it raises an *invalid preconditions* failure;
- **error:** when a service request is sent to the robot control architecture, an execution report is expected; this report can be either a success report or a failure report, in which case an *error* is raised;
- **invalid effects:** when a service reports successfully, the **`MissionManager`** checks that the system state is consistent with the expected effects of the service; otherwise, it raises an *invalid effects* failure;
- **timeout:** in the case when a maximal duration is defined for the service, the **`MissionManager`** raises a *timeout* failure if the service has not reported before the duration expired.

When one of these failures is raised, the robot plan must be repaired.

E. Plan Repair

The basic idea of plan repair in HiDDeN is that when a task fails, we have to replace this task (and only this one) by an alternative task that allows to achieve the mission. If such an alternative does not exist, we take advantage of the hierarchical structure of the iHTN plan to re-plan only a sub-tree of the plan that has a valid alternative, not the entire mission. More precisely, when a task t fails, we ask a planner to solve a new local planning problem. This planner could be the same as the initial planner, or a dedicated planner, as long as it provides a hierarchical plan. This new local planning problem is generated as a *planning request* by the **`PlannerManager`** according to the current system state and the failure status.

A *planning request* is composed of a *domain* (actions that can be used to modify the state of the world) and a *problem* (one instance of the state of the world, i.e the initial and goal states). We define D our initial domain, and $P = (s, g)$ our initial problem, with s the initial state of the world and g the goal state to achieve. During the repair process, we can define the *planning request* $\mathcal{R} = (\mathcal{D}', \mathcal{P}')$ such as:

- $\mathcal{D}'$ the updated domain taking into account the failure;
- $\mathcal{P}' = (s', g')$ the updated problem, with s' the current state of the system (available through the **`DataManager`**) and g' the new state goal to achieve (i.e. the abstract task that must be replanned).

The domain must be modified to integrate that the faulty elementary task is not applicable any more (in the current situation). We first define $\mathcal{D}'$ by adding the negation of current state s' into the preconditions of the faulty elementary task. This prevents the planner to propose a repair result as the one that has just failed. More elaborate definitions of $\mathcal{D}'$ are discussed in the future works.

This *planning request* is then sent to the embedded planner. If the planner returns a new plan, this plan then replaces the failing task t in the iHTN. Otherwise, we have to go up in the iHTN structure to repair the parent task in an iterative manner, until we find a valid alternative or we reach the root task (Fig. 4).

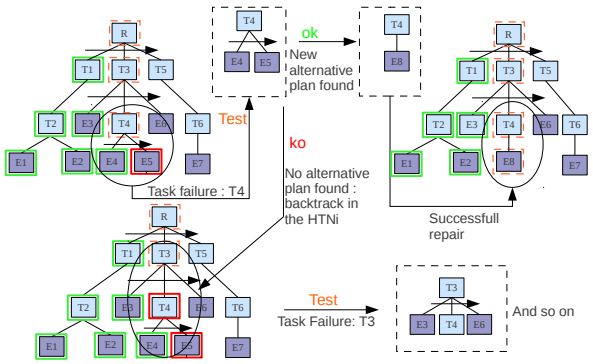


Fig. 4: The repair process: elementary task E_5 fails, which implies a repair of task T_4 (top, left). If a new plan is found, the iHTN branch is replaced (top, right); otherwise, the repair process is applied to T_3 (bottom).

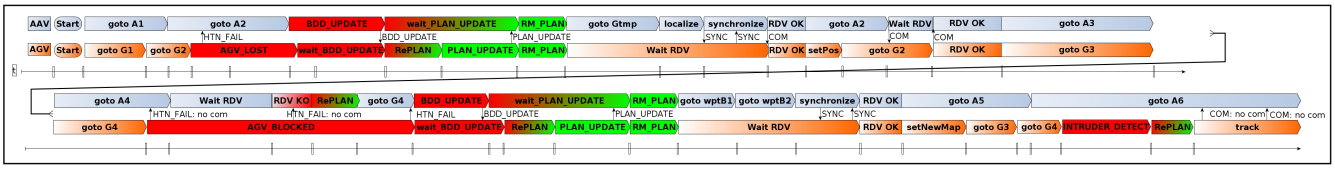


Fig. 5: Example of resulting timelines.

F. Synchronization for Plan Repair

When repairing a task t, we must take into account that this repair will affect all the robots involved in the achievement of t (where t is an elementary task or an abstract task that "failed" because of the bottom-up repair process). The repair of a multi-robot task will therefore need a synchronization among the involved robots, carried out by their **`CoopMessenger`** modules. This synchronization is enforced by a protocol described into predefined iHTN schemes. This process is illustrated with the sequence diagram of Fig. 6, described below. In this example, robot R1 has to repair the failing task E4. As no local plan is available to repair E4, the repair process goes up to the parent task T2, that is a task involving both R1 and R2. We then apply a cooperative repair process, that is modeled as an iHTN and inserted into the current plan, replacing E4. The execution of this iHTN is emphasized by the RepairingPattern block of the sequence diagram: the HTN-FAIL call warns R2 that a repair is needed; in return, R2 sends its Koper database so that R1 can update the current state (message BDD-UPDATE); R1 proceeds to the repair of task T2 (resulting in a new task T6); and R1 sends T6 to R2 (message PLAN-UPDATE)[3]. Then T6 is inserted into the plan, replacing T2, and the plan execution can go on for both robots.

If the HTN-FAIL call fails, meaning the communication between R1 and R2 is not possible at the moment, a similar CommunicationPattern is applied to help R1 establish the communication with R2. This pattern tries several predefined recovering strategies: communication relaying via other robots, area exploration to find R2, etc.

V. EXPERIMENTATION AND EVALUATION

We have experimented our HiDDeN architecture on a real mission involving two robots: an AAV (an autonomous rotorcraft, Fig. 7a) and an AGV (a custom rover based on a Segway base, Fig. 7b). The two robots have to explore a rural area in order to find a possible intruder. Several executions of this mission permited to test and evaluate the HiDDeN architecture in several cases: (a) with different `goto` failures, resulting in several repairs (e.g., one based on relative vision-based localization, another one based on laser-based mapping); (b) varying communication failure cases (no failure, failure in a planned communication, failure in the repairing pattern); (c) detection of an intruder involving a plan repair. In all these cases, HiDDeN has succeeded in managing the plan execution and the repair process.

[3]Note that the plan distribution process described in section IV-B ensures that the task T2 will be present in the local plans of both R1 and R2.

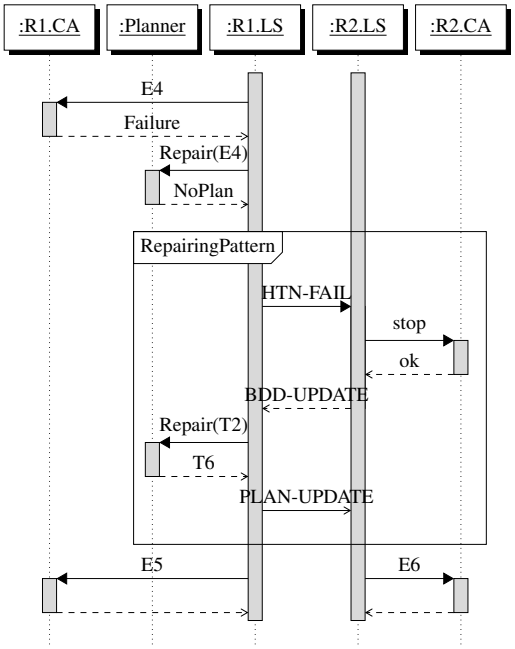
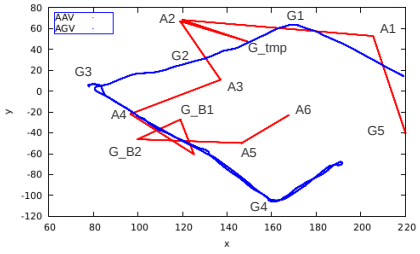


Fig. 6: Sequence diagram of synchronization of two robots for a plan repair. R.CA represents the Control Architecture of robot R, R.LS the Local Supervisor of robot R.

(a)

(b)



(c)

Fig. 7: Experimentation vehicles and result samples.

Trajectories for a particular mission run are presented in Fig. 7c, and the associated timeline showing motion and communication actions is depicted on Fig. 5. The plan mission consists in reaching (and exploring) eleven different waypoints of the area: six for the AAV (noted A1 to A6 on Fig. 7c) and five for the AGV (noted G1 to G5 on Fig. 7c). Four rendez-vous between the robots are also expected. The robots encountered the three following failures during the plan execution:

1) The AGV fails while reaching its second waypoint (`goto G2` error), due to its individual localization

process which is only vision based; AGV's *local supervisor* tries first a local plan repair, but since it is lost, the AGV is unable to find a new path on its own; then, going up in the plan hierarchy, a new plan is computed (applying the RepairingPattern): the AAV has to reach G_tmp, the last known position of the AGV, provided by the data knowledge of the AGV, and has to localize the ground robot; the plan execution can then continue as initially planed.

2) Trying to reach $G4$, the AGV is blocked by an unexpected obstacle; AGV's *local supervisor* tries first a plan repair attempt which fails (no alternative plan can be found in the current AGV map); then, a team repair must be done; this time, communication is not available between the robots: the communication recovery pattern applied in this experiment was for the AGV to wait until the AAV finds it; meanwhile, the AAV has reached a rendez-vous point and a failure is detected as the communication was not possible (the AGV is not there); the communication recovery pattern for the AAV was to explore the area until a communication with the AGV is possible; once the communication link is recovered, the RepairingPattern can be applied, leading to a new plan: the AAV maps the area surrounding the AGV position, then sends it to the AGV that can now find a new path to join its next waypoint.
3) The AGV finally detects an intruder, and its plan is repaired into a tracking action while trying to inform the AAV of its movement.

VI. CONCLUSIONS

In this paper, we have presented a high level distributed architecture, HiDDeN, for multi-robot systems composed of robots that are already autonomous but not necessarily built in a multi-robot concern. The system is composed of local supervisors that are embedded on each robot, and interfaced with their local architecture. A complete fault detection management is proposed, which determines when a recovery process is needed and increases as much as possible fault tolerance during the mission execution. The main contribution of our work is that we use as little communication as possible, by first distributing the global mission plan to each robot (removing unneeded tasks and adding necessary communication tasks), and second repairing the plan as locally as possible, while communicating only with the robots involved in the repair process.

An improvement would concern the definition of a local planning problem in case of failure. For now, we just update the current state, and avoid using the same action again by modifying its preconditions. We are currently considering the possibility to apply, on-line, a learning algorithm in order to update the planning and environment models all along the mission execution. We also want to explore the possibility of having conditional plans, i.e iHTNs where different methods are available and are instantiated, to allow an adaptation of the plan execution to the dynamic environment without a systematic repair. In these plans, the execution may also be improved by heuristics choices in the tasks method where preconditions are satisfied, improving efficiency according to the current observed environment state.

ACKNOWLEDGMENT

This work has been partially supported by the DGA funded Action project (`http://action.onera.fr`) and the ANCHORS project (`http://www.anchors-project.eu`), funded by the German Federal Ministry of Education and Research (BMBF) and the French National Research Agency (ANR).

REFERENCES

[1] I. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "CLARAty: Challenges and steps toward reusable robotic software," in *Int. Journal of Advanced Robotic Systems*, vol. 3, no. 1, 2006.
[2] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: Planning at the core of autonomous reactive agents," in *Int. NASA Workshop on Planning and Scheduling for Space*, 2002.
[3] S. Lemai and F. Ingrand, "Interleaving temporal planning and execution in robotics domains," in *AAAI*, San Jose, CA, USA, 2004.
[4] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A deliberative architecture for AUV control," in *ICRA*, Pasadena, CA, USA, 2008.
[5] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *JAAMAS*, vol. 19, no. 3, 2009.
[6] F. Teichteil-Königsbuch, C. Lesire, and G. Infantes, "A generic framework for anytime execution-driven planning in robotics," in *ICRA*, Shanghai, China, 2011.
[7] R. Vidal, O. Shakernia, H. Kim, D. Shim, and S. Sastry, "Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation," *Trans. on Robotics and Automation*, vol. 18, 2002.
[8] L. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation," *Trans. on Robotics and Autom.*, vol. 14, 1998.
[9] L. Chaimowicz, T. Sugar, V. Kumar, and M. Campos, "An architecture for tightly coupled multi-robot cooperation," in *ICRA*, Seoul, Korea, 2001.
[10] M. Tambe, "Towards flexible teamwork," *JAIR*, vol. 7, 1997.
[11] M. Paolucci, O. Shehory, and K. Sycara, "Interleaving planning and execution in a multiagent team planning environment," *Linköping Elec. Articles in CIS*, vol. 5, no. 18, 2000.
[12] R. Smith, "The Contract Net Protocol: high-level communication and control in a distributed problem solver," *IEEE Trans. on Computers*, vol. 29, no. 12, 1980.
[13] M. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Eng. Practice*, vol. 15, no. 4, 2007.
[14] N. Atay and O. Bayazit, "Emergent task allocation for mobile robots," in *RSS*, Atlanta, GA, USA, 2007.
[15] S. Joyeux, R. Alami, and S. Lacroix, "A plan manager for multi-robot systems," in *FSRS*, Chamonix, France, 2007.
[16] R. Micalizio and P. Torasso, "Supervision and diagnosis of joint actions in multi-agent plans," in *AAMAS*, Estoril, Portugal, 2008.
[17] C. Sotzing, N. Johnson, and D. Lane, "Improving multi-AUV coordination with hierarchical blackboard-based plan representation," in *PlanSIG*, Edinburgh, UK, 2008.
[18] F. Legras and C. Tessier, "LOTTO: group formation by overhearing in large teams," in *AAMAS*, Melbourne, Australia, 2003.
[19] T. Gateau, C. Lesire, and M. Barbier, "Knowledge base for planning, execution and plan repair," in *ICAPS PlanEx Workshop*, Altibaia, Brasil, 2012.
[20] K. Erol, J. Hendler, and D. Nau, "HTN planning: complexity and expressivity," in *AAAI*, Seattle, WA, USA, 1994.
[21] J. Dix, H. Muñoz-Avila, D. Nau, and L. Zhang, "IMPACTing SHOP: putting an AI planner into a multi-agent environment," *Annals of Mathematics and AI*, vol. 37, no. 4, 2003.