Knowledge Base for Planning, Execution and Plan Repair

Thibault Gateau, Gaëtan Severac, Charles Lesire, Magali Barbier, Eric Bensana
Onera - The French Aerospace Lab - F-31055, Toulouse, France
firstname.lastname@onera.fr

Abstract

Future robotic missions will involve numerous and heterogeneous autonomous robots. The management of these robots, in order to achieve a mission, requires interoperability, not only at low-level communication protocols and operational HMIs, but also at the decision level, to ensure a robust plan execution and repair framework. Through a formal representation of multi-robot knowledge using an ontology, we show how such a model is of high interest in autonomous mission monitoring, as it provides relevant information all along the mission execution and how it helps to define communication and execution protocols from initial planning computation to plan repair in a multi-robot context.

Introduction

Planning, executing and repairing a mission plan is quite easy... in a simulated, deterministic, well-defined and closed environment. In real world, things are more challenging when different autonomous vehicles must achieve together a team mission.

Robots have already reached a high level of autonomy. They are used in many different situations, and most of the time, their autonomy settles on a well-defined embedded software architecture. Many architectures have been proposed in the literature (McGann et al. 2008; Doherty, Kvarnström, and Heintz 2009; Teichteil-Königsbuch, Lesire, and Infantes 2011) and are currently in use for complex real-world situations where autonomous robots must manage the execution of their own tasks.

The advance of the work in this field has led researchers wonder for many years to what extent robots would be able to be integrated into a team, not only concerning one or more human operators, but most importantly other autonomous and heterogeneous robots, with complementary functionalities. Such situations will indeed become more and more common, as robotic missions are becoming more and more complex, looking for more and more autonomy. We can already suppose that robots will be deployed incrementally, because of the cost and time required for setting up such technologies. For instance, in spatial exploration, robots (rovers, satellites) are deployed one after the other, and missions can last a couple of decades. It is reasonable to consider that the number of robot collaborations will increase in the future, for the benefits of science (Visentin 2007).

Robots will then be intrinsically heterogeneous, as they will be developed gradually, and probably by different builders. Hence, their embedded architecture may even 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, without re-designing all their control architecture. But, even assuming that this interoperability issue is solved, we are then confronted with the existing gap between planning and executing a real mission in nearly complete autonomy. Having a team of autonomous vehicles achieve a mission needs first the computation of a mission plan. Then, in order to execute this plan and monitor the execution, a *protocol* must be defined to interface "plan actions" to "robot's tasks". Finally, while executing such a plan, failures *will* occur, as the autonomous vehicles will have to face environment disturbances whatever the assumptions made in the plan are. So, plan-repair (or complete re-plan) is compulsory in achieving mission goals. In order to generate *on-line* a relevant planning model, information about the current state must be gathered from the vehicles. Again, interfaces are required between the individual heterogeneous architectures and the planning system.

In this paper, we propose a formalization of knowledge representation in multi-robot autonomous missions. This formalization aims to lay the basis for interface needs in planning, plan execution and plan repair for a team of autonomous vehicles.

State of the art

Mission execution involving several autonomous vehicles has been widely studied. We examine in the following state of the art how links between planning and execution are considered, and how on-line replanning activities are managed.

The BErkeley AeRobot (BEAR) project[1] studies multi-agent probabilistic pursuit-evasion games with heterogeneous robots. In this project, (Vidal et al. 2002) developed a distributed, hybrid and hierarchical architecture system, in order to manage partial knowledge of states among the team. They take into account a dynamic environment, het-

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

erogeneous agents, faults on robots, and sensor imperfections. Contrary to the *sense-model-plan-act* architectures, robot's dynamics is taken into account at the 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 numerous, which leads to a complex adaptation process to re-use existing mono-agent architectures. The environment knowledge is well defined for feeding a tactical and a strategy planner, but specific to the current mission.

The ALLIANCE architecture (Parker 1998) 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. The robots are of the same type (but with different abilities) and are controlled by the same architecture. In the same way, (Chaimowicz et al. 2001) 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 of the robot control manager. The system is designed for a specific mission.

In Hierarchical Task Network-based approaches (Nau et al. 2003), the planning problem is modeled in a hierarchical way. The Retsina architecture (Paolucci, Shehory, and Sycara 2000) manages the execution of such a hierarchical plan: a dynamic list of objectives is stored as a priority queue, and each agent selects a task to be executed. Thanks to their ability to achieve planning tasks interleaved with executive tasks, the agents are thus highly adaptable to the environment. Concrete interaction with a real environment is not yet described, and they seem to remain only in a planning point of view. Partial replanning is addressed by inserting new tasks into the priority queue in order to have a new resolve of the partial planning problem. Nevertheless, they do not manipulate an explicit global team plan, making it difficult to reason at team level and to manage team organization. (Fazil Ayan et al. 2007) emphasizes local repair in such a hierarchical plan structure in a mono-agent context. A dependency graph between HTN tasks is built, that allows the replanning module to know which tasks need to be replanned in case of failure. Hence, they avoid recomputing well on-going parts of the plan. Considered data is adapted for re-planning processes, but they are not dealing with real robotic missions.

(Sotzing, Johnson, and Lane 2008) explicitly deal with knowledge representation and update in multiple AUV (Autonomous Underwater Vehicle) operations. Mission execution and multi-vehicle coordination is supported by BI-IMAPS (Blackboard Integrated Implicit Multi-Agent Planning Strategy). All vehicles have a complete copy of the BI-IMAPS plan, so the vehicles' actions can be predicted when there is no communication. When communication is anew available, robots refresh the knowledge coming from other vehicles. Communication management is a well known issue in multi-robot architectures. Furthermore, entities which are interacting must agree on the communication protocol, and the exchanged type of data. An implicit agreement about data is often considered in robotic team mission, but not formally described. When the same architecture is controlling all the robot team, (Tambe 1997) points out that *the more the team is flexible and robust, the higher communication load is required*. Thus, he emphasizes the importance of communication management in team mission execution, as this represents the most critical resource and may lead to a mission failure. In robotic space exploration missions, mainly because of communication constraints, it is impossible to consider a fine coordination of a set of vehicles from a mission control based on Earth. Part of this coordination must therefore be done on site, by the robotic agents themselves. Consequently, local communications between vehicles and a significant degree of autonomy in the vehicles' interactions will be assumed. Different space agencies (ESA, NASA...) already started to evoke future needs concerning standards and interoperability (Tramutola and Martelli 2010; Chien et al. 2006; Merri et al. 2002). Meanwhile communication standards and procedures are already used by (Kazz and Greenberg 2002), but at the equipement level.

In the literature of heterogeneous multi-robots missions, we can notice that protocol formalization is not satisfying. First, regarding protocols for plan execution and repair, ad-hoc interfaces (between planners and vehicles) are implicitly used to translate the mission plan into a language understandable by an autonomous vehicle. Plan repair is often considered from a planning point of view, not from the point of view of the generation process of the data that must be provided to the planner, and coming from heterogeneous sources. Besides, regarding protocols for multi-robot communication, a common assumption is made that all autonomous vehicles of a same team are implicitly working with the same standards.

For these reasons, we propose to formalize the common knowledge involved in (re)planning and executive phases. We describe first our knowledge representation, based on an ontology describing what the vehicles are able to achieve, and how interactions are managed. Afterwards, we present the use of an instantiated version of this ontology for mission planning, plan execution, and on-line plan repair.

Knowledge Representation

Ontologies in robotics

An ontology is a common representation of a specific domain that allows different individuals to share concepts and rich relations (Baclawski and Simeqi 2001).

In robotics, ontologies are used to specify and conceptualize a knowledge accepted by a community, using a formal description to be machine-readable, shareable (Sellami et al. 2011) and to reason over that knowledge to infer additional information (Schlenoff and Messina 2005). Ontologies offer significant interests to multi-agent systems such as interoperability (between agents and with other systems in heterogeneous environments), re-usability and support for multi-agent system development activities (Tran and Low 2008).

Ontologies must be used with care (Lortal, Dhouib, and Gérard 2011): an ontology (specification) must not be confused with a knowledge base (which actually includes

knowledge). In the following, an *instantiated ontology* will refer to the knowledge base. Ontologies depend highly on their builders, and allow sharing of information between agents (Deplanques et al. 1996).

Ontologies are already being used in many different projects. In the context of Web Service the system of (Sirin et al. 2004) executes a plan computed with SHOP2 (Nau et al. 2003) over the web. It is able to execute information-providing Web Service during the planning process. We must note that they provide a sound and complete algorithm to translate OWL-S[2] service description into a SHOP2 domain. The Robot Earth European project (Waibel et al. 2011) aims at representing a world wide database repository where robots can share information about their experiences, with abstraction to their hardware specificities. But it is a starting project, without exploitable results yet, and it deals more about environment knowledge representation and sharing. In the Proteus project (Lortal, Dhouib, and Gérard 2011), complex ontologies are used for scientific knowledge transfer between different robotics communities. However, the developed ontology cannot be used directly for code generation and exploitation: authors have to perform semi-automatic transformation from the ontology to a UML representation. The ontology is also quite specific to their application scenarios problems. For space applications, the SWAMO NASA project (Witt et al. 2008) uses ontology as a prototyping method to provide standard interfaces to access different mission resources (sensors, agent capabilities...). In a similar approach, the A3ME (Herzog, Jacobi, and Buchmann 2008) ontology defines heterogeneous mobile devices, to allow communication interoperability. (Schlenoff and Messina 2005) has also worked on robots' capabilities representation in the context of urban search and rescue missions.

Those studies are very interesting and represent a starting point for our work, but the proposed ontologies are there at a lower level of knowledge representation. They focus more on the description of the capacities of mobile agents than on high level services representation for autonomous agents, as we aim to do. Because of these limitations, we choose to define a new ontology.

Formal robot description

The information we need to model in our ontology must take into consideration (1) robots involved in the mission and the information they are able to provide, (2) a description of the environment, and (3) a description of the mission goals. We must note that many ontologies already define services in the Web Service domain (e.g. OWL-S, or WSDL[3]). Our goal is to use a simple, effective ontology to support plan execution and repair. Different kinds of languages exist to model ontologies in computer readable text format. Our choice naturally falls on the most used in the domain of Artificial Intelligence which is the Web Ontology Language (OWL), defined by the World Wide Web Consortium[4] (W3C). OWL is an XML-based format for writing ontologies in Description Logic (DL). Many tools have been developed to design and manipulate ontologies. We have used Protégé[5] to design our ontology which is partially depicted in Fig. 1 and explained below.

To define this ontology, we were inspired by real robots execution management that we have been experimenting before and on the paradigm of programming by contract for the services that a robot can provide (Brugali and Scandurra 2009).

Robot Services Robots are classically defined by the services that they provide, and how agents can be interfaced with them (Brugali and Scandurra 2009). A `Service` is a task or an activity that a robot (or a software agent) can achieve in its environment (including what its sensors capabilities are able to get at a high level, e.g. target detection, rock analysis...). A service must implement a `Contract` (`SerContract`). It may need input parameters (`ConNeed`) to be executed and may return output parameters (`ConReturn`). It may be associated with specifications such as preconditions (`ConRequire`), postconditions (`ConEnsure`) or invariants (`ConInvariant`).

A `Service` is defined by its name (`SerName`) and a description (`SerDescription`). `SerAccess` describes how a distant entity (distant robot, team executor manager...) can call the service of the robot, to make the robot achieve it. `SerType` distinguishes *direct* services that can be directly called (goto action, take a picture, process data, etc.), and an execution result from them is expected, and *daemons*, which are running as background tasks and may return particular events (target detection, communication link broken, ...).

Robot Internal Variables A description of the knowledge manipulated by the robot is also required, all the more as these values are involved during the planning process. For example, if the robot has a GPS, it is a good assumption to have a $RobotPositionGPS$ variable. A variable, in the ontology, has classical entities: an identifier (`VarId`), a type (`VarType`) and optionally an informal text description (`VarDescription`). `VarGetter` and `VarSetter` are comparable to classical accessors in Object Oriented Programming, meaning the way to access the value of the variable, and a potential way to modify its value. The localization (place where the value is stored) of the variable (`VarLoc`) is defined, which is indirectly linked with the `VarInfluencers`. This last entity indicates which elements (vehicles, operators, variables...) may have an influence on the variable, and which elements must be observed during the execution process to detect particular evolution of the variable value. For instance, a robot "controls" its own position variable, other robots cannot "influence" this variable but can only read it.

Environment, Models and Relations Environment variables (areas, objects of interest, ...) and goals are also modeled as `Variable` entities. In addition to traditional variable type, we use a particular type ($Model$) corresponding

[2] `http://www.w3.org/Submission/OWL-S/`

[3] `http://www.w3.org/TR/wsdl`

[4] `http://www.w3.org/standards/techs/owl`

[5] `http://protege.stanford.edu/`

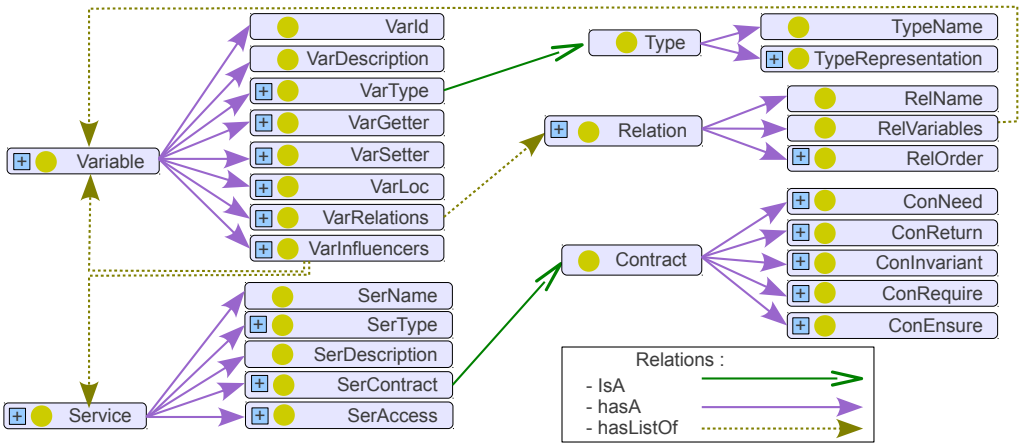

Figure 1: Partial view of the developed ontology, free adaptation from Protégé-Ontograf viewer

to robotic models: they allow to link some variables with external models, such as estimations of communication availability, estimations of the fuel consumption of a robot, etc.

`Relation` entities allow to interconnect variables when needed, and to add particular characteristics to them. For instance, a zone $z1$ can be a sub-zone of a zone $z2$, without creating new specific types for these variables.

With this ontology, we have a description of what the vehicles are able to achieve, how interactions are managed, and in what kind of environment they are dived into. Finally, this ontology regroups both planning and execution information.

Ontology in practice

The ontology described before is meant to be instantiated. The ontology provides the interface between both execution and plan-repair processes, and between the execution process and the vehicle's architecture.

In the following we present how to use the instantiated ontology, named KOPER (KnOwledge base for Planning, Execution and Repair) for (1) generation of planning model, (2) plan execution monitoring and (3) on-line repair. In each case, we discuss a generic use-case of the ontology, and then give a more concrete example of our own HTN-based architecture.

Planning Model Generation

We discuss here a mean to generate from the KOPER ontology the required information destined for a planner, which is then able to compute a mission plan.

Generic Planning Model Generating a complete planning model from an external data base without human intervention is far from being easy when confronted with real world missions, unless these real world missions are already fully described in a planning language. For that, we propose an automatic generation of the domain file from a KOPER instantiated ontology. While such a generation pattern can be generalized to almost all the planning languages, we illustrate this generation on the classical PDDL language (Drew McDermott et al. 1998). In PDDL, a planning model is made of a planning domain (that describes predicates and actions) and a planning problem (that instantiates domain variables, and defines the initial and goal states).

PDDL Domain: A PDDL domain is made of a set of $actions$. A PDDL action is described by a name, some parameters, a precondition and an effect. We can use Model to Model (M2M) transformation to generate the corresponding PDDL action from each `Service` of the instantiated ontology. `SerName` corresponds to the PDDL action name. `ConNeed` provides the PDDL action parameters information, `ConRequire` corresponds to the action preconditions, while `ConEnsures` and `ConInvariant` determine the action effects. Some `Variable` of the instantiated ontology can be involved in the domain generation, since they can be "influenced" by services. Listing 1 presents the instantiation of a *goto* Service provided by an instantiated $Vehicle$, an AAV (Autonomous Aerial Vehicle). The corresponding PDDL action is shown in listing 2, in which the $?v$ variable corresponds to a $Vehicle$.

PDDL Problem: A PDDL problem is made of a set of objects, an initial state and a goal. We use again a M2M transformation to generate the PDDL problem by considering entities `Variable` and `Relation`. The set of PDDL objects is generated from the `VarId` and the `VarType` of each `Variable`. The initial state description requires an exhaustive cover of the `Variable` values, accessed with combined `VarGetter` and `VarLoc` information. The `Variable` planning value is then formatted depending on the `VarType` documentation. The `Relation` entities are then interpreted, since written variables may be inter-linked. Finally, the goal is similarly generated, corresponding in our instantiated ontology to a particular `Variable`. Listing 3 presents a variable definition in the ontology and some relations, and listing 4 the generated PDDL problem.

HTN Planning Model in HiDDeN HiDDeN (Gateau, Lesire, and Barbier 2010) is a High level Distributed DecisioN layer that monitors execution and plan repair for a

```
<Service>
 <SerName>goto</SerName>
 <SerType>DirectTask</SerType>
 <SerDescription>
   The vehicle moves to the ?wptTo location
 </SerDescription>
 <SerContract>
  <ConNeed>
   <Arg>
    <ArgName>?wptFrom</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
   <Arg>
    <ArgName>?wptTo</ArgName>
    <ArgType>Waypoint</ArgType>
   </Arg>
  </ConNeed>
  <ConReturn>
   <Arg>
    <ArgName>?msgReturn</ArgName>
    <ArgType>SimpleReturn</ArgType>
   </Arg>
  </ConReturn>
  <ConRequire>
    <Cond>(this.pos == ?wptFrom)</Cond>
    <Cond>(this.st != this.onGround)</Cond>
  </ConRequire>
  <ConEnsure>
    <Cond>(this.pos == ?wptTo)</Cond>
    <Cond>(this.pos != ?wptFrom)</Cond>
  </ConEnsure>
  <ConInvariant>
   <Cond>(this.enoughFuel)</Cond>
  </ConInvariant>
 </SerContract>
 <SerAccess
  ComBase="Socket"
  OutPort="60100"
  ParamUsed="?wptTo"
  OutMsgFormat="SocketMsg('GOTO'+Waypoint)"
  InPort="60101"
  InMsgFormat="SocketMsg(SimpleReturn)"/>
</Service>
```

Listing 1: Example of possible "goto" service contract for an AAV in XML representation transformed from the OWL description ; the *this* keywords refers to the service owner

```
(:action goto
 :parameters (?wptFrom ?wptTo - Waypoint ?v
     - Vehicle)
 :duration (= ?duration (
    function_gotoDuration ?wptFrom ?wptTo ?
    v))
 :condition ((enoughFuel ?v)(position ?v ?
    wptFrom)(not (onGround ?v)))
 :effect (and((not(position ?v ?wptFrom))(
    position ?v ?wptTo))))
```

Listing 2: "goto action" in pddl style

```
<Variable>
 <VarId>aav1Position</VarId>
 <VarType>Waypoint</VarType>
 <VarGetter>Service::getGPSPoint</VarGetter>
 <VarSetter>none</VarSetter>
 <VarLoc>local</VarLoc>
 <VarRelations>
  <RelName>aav1PositionRelation</RelName>
  <RelVariables>
   <VarId>this</VarId>
  </RelVariables>
  <RelOrder>simple</RelOrder>
 </VarRelations>
 <VarInfluencers
  var="this"
  ser="Service::goto"/>
</Variable>
```

Listing 3: Example of a variable in XML representation transformed from the OWL description; the *this* keywords refers to the variable owner

```
(:objects
 aav1 - Vehicle
 aav1Position - Waypoint
 wpt2 - Waypoint)
(:init
 (aav1 at aav1Position))
(:goal
 (aav1 at wpt2))
```

Listing 4: Partial PDDL problem: $aav1Position$ is described in Listing 3. Variable $wpt2$ comes from similar OWL descriptions, alike a $goalVariable$ which precises that $aav1$ must reach waypoint $wpt2$; $aav1$ is an instantiated vehicle; a relation exists between $aav1$ and $aav1Position$ (`RelVariables` field in Listing 3) that leads to the generation of the initial state line.

team of autonomous robots. The underlying planning model is based on HTNs (Nau et al. 2003). 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. Theoretically, if the recipe provided by a method, picked out amongst the possible methods of the task, is followed, then the objectives of the task will be achieved. The interests of HTN are their flexibility, their hierarchical structure and their convenient modeling of human knowledge. Besides, multi-agent planners have been specifically developed to deal with HTN formalism (Dix et al. 2003).

Regarding our instantiated ontology, we can add the HTN structure (i.e. the abstract tasks corresponding to human expertise) either in the knowledge representation as "high level" `Service` entities, or directly in the generated planning model. We chose the second solution as human expertise is in a sense more related to the planning model (an ex-

pert can add abstraction, but also domain-specific heuristics or constraints) than to robot or mission description.

One of the modeled missions is a mine hunting scenario involving an AAV and an AUV. The instantiation of the KOPER ontology for this scenario is composed of 18 robots' services. 34 variables are described (including goals and models), with 10 relations. The generated HTN domain contains 18 elementary tasks, increasing to 23 when we add HTN abstraction from a human expert. The generated HTN problem contains 26 objects, and 92 conditions in the initial state.

We have solved this problem using SHOP2, and our HTN planner. Both planners generate a plan containing 48 abstract tasks and 60 elementary tasks.

Plan Execution

After generating a planning model from the KOPER instantiated ontology, we can use an off-the-shelf planner to compute a mission plan. In this section, we describe how the KOPER ontology can assist in monitoring the plan execution, assuming that a mission plan has been computed.

Generic Plan Execution Plan execution consists in executing each action written in the plan by the concerned robot(s). Since robots have specific protocols for external interaction with them, the plan action must be translated according to this protocol, which describes how to make the robot execute the desired action. Thanks to the KOPER instantiated ontology, the action encountered by the execution manager is automatically mapped with this communication protocol, in particular with the `SerAccess`. In fact, `SerAccess` precises the type of communication mode, the associated configuration and also the data format and content that must be sent. When a robot executes an action, a report is expected to indicate whether the task has been completed or not. Once again the format and the manner are described in the KOPER instantiated ontology through `ConReturn` and `SerAccess`. The service contract (`SerContract`) allows failure detection means during the execution process. The execution manager may check that: (a) action's precondition (`ConRequire`) is true before executing, (b) action's actual effects are consistent with the action model (`ConEnsure`) and (c) action's invariants remain true during the execution.In fact, KOPER provides a mapping between execution variables and planning (logical) variables.

For instance, we take the action "goto wpt1 wpt2 aav1" from a PDDL planner output plan: AAV *aav1* must go from waypoint *wpt1* to *wpt2*. The execution manager finds in the KOPER instantiated ontology the corresponding service with the `SerName` "goto" (listing 1). It is then aware that communication with the local architecture is based on sockets (line 38), with a given port and a message format to send the execution order to the robot. The "Waypoint" parameter (from the `ConNeed` field) is passed to the robot, after getting its value through the `VarGetter` function. Finally, the plan execution process can send a socket message formatted into understandable data to the robot architecture. When this message has been sent, the execution manager waits for a "SimpleReturn" type message, on another socket, that corresponds to waiting for a boolean report (`true` if the robot has reached $wpt2$, `false` otherwise).

Daemon services are not triggered by the execution process: they are active all along the mission and can send information to the execution process according to the protocol described in the ontology. Such information can correspond to specific observations (e.g., a detected target) and most of the time will result in a plan adaptation or repair.

HTN-based Plan Execution in HiDDeN In HiDDeN, we directly use the HTN structure of the mission plan (hence resulting in an instantiated HTN, see (Gateau, Lesire, and Barbier 2010) for details) as the core model of plan execution monitoring.

HiDDeN is a distributed architecture that settles on the existing robots' architectures, takes care of what specific action has to be executed by which robot, and controls the coordination between robots (Fig.2). Each local supervisor is then able to communicate with all the rest of the team, depending on communication availability.

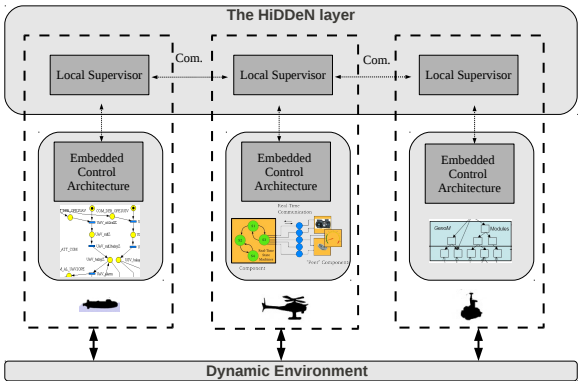

Figure 2: The HiDDeN layer monitoring a team of heterogeneous robots.

The main purpose of the local supervisor is to provide the action that the robot has to execute, depending on the local plan. The local HTN tree plan is followed in a depth first and left first manner. When an elementary task (a leaf) is reached, the corresponding action execution is requested to the local architecture (as explained in Generic Plan Execution). For instance, in Fig. 3, the root task to achieve is R. The execution manager enters into task $T1$, itself decomposed into $T2$. $E1$ must then be executed first by the robot. If this execution is successful (i.e. the report is `true`), $E2$ must then be executed. At the end of $E2$, $T1$ is considered as done. Afterwards, the execution manager enters into $T3$, and so on.

Failure detection in HiDDeN is based on monitoring actions' contracts (as described in Generic Plan Execution), analysing returned values after the execution ($ConReturn$), and timeouts events.[6].

[6]To avoid freezing, the execution engine arms a timer when running an action. When the timer has expired, the action is interrupted

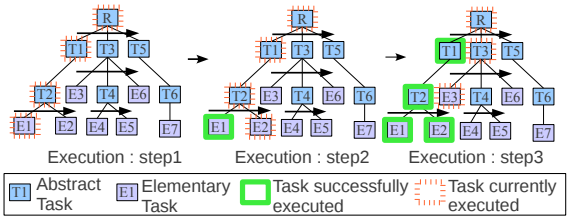

Figure 3: Execution of an instantiated HTN: the elementary execution order is $E_1, E_2, E_3, E_4, E_5, E_6, E_7$.

Regarding multi-robot coordination, HiDDeN plans are computed in order to ensure regular rendezvous between vehicles all along the mission. These rendezvous are 4D space (location and time) where two vehicles are able to communicate. During these rendezvous, the distributed supervisors of HiDDeN will update their team knowledge. This update is based on the KOPER instantiated ontology, that defines the team variables, their `varInfluencers` (which own the actual value of the data) and variable getters and setters (to access the data and store it).

Plan Repair

While executing the plan mission, the autonomous team will encounter failures. The team must then find an alternative plan to achieve the mission. For that, thanks to the KOPER instantiated ontology, an up-to-date planning model can be provided to an embedded planner, which can repair the current plan or compute a new plan.

Generic Plan Repair We consider here that plan repair is the responsibility of the execution process, in the sense that it consists in defining a new planning model corresponding to the part of the plan that needs to be repaired.

This process is hence very similar to the initial generation of planning domain and problem. However, variables' values have been modified during the execution process (e.g. the position of the vehicle has changed after a "goto" action). Updating these variable values is done through the KOPER instantiated ontology thanks to `VarLoc` and `VarGetter`.

The services list description can also change (e.g a service has an unexpected result and is "desactivated", or is not provided any more by a robot, a robot is out of service...), impacting then a different planning domain generation.

HTN Plan Repair in HiDDeN Plan repair in HiDDeN is globally done the same way as in Generic Plan Repair: by defining the local planning model to solve. However, we extensively use the HTN hierarchy in this process: once an elementary action failed, we first try to repair it (by defining a planning model corresponding to performing this action). If this repair succeeds, we replace the task and go on with execution. If the repair fails (either because no solution to the local problem can be found, or because the problem cannot even be defined due to unavailable data), HiDDeN goes up in the hierarchy and tries to repair the task just above the previous one (i.e. the abstract task containing the previous task), and so on until a repair plan is computed (Fig. 4).

As in our scenarios communication between the vehicles is not always available and is subject to disturbances, the KOPER instantiated ontology is very useful in providing the localization of variables that must be accessed to define the new planning problem. It limits communication to the required ones and allows to determine which communication links must be established to repair the plan and achieve the mission.

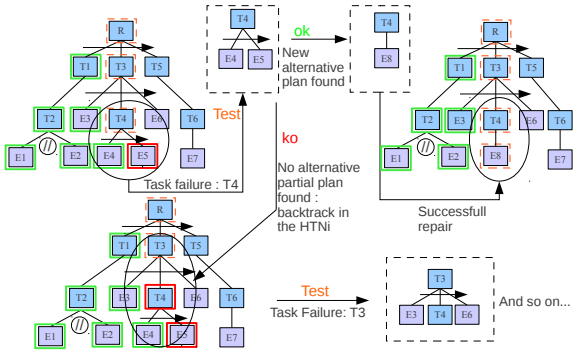

Figure 4: The repair process : elementary task E_5 fails, which implies a repair of task T_4 (top, left). If a new plan is available, the corresponding instanciated HTN branch is replaced (top, right); else, the first hierarchically superior task of T_4, here T_3, is repaired (bottom).

Conclusion

In this paper, we have defined an ontology, KOPER, to represent the necessary knowledge used during planning, plan execution and plan repair for a team of heterogeneous autonomous robots. KOPER is based on a description of robots' services with associated parameters and contracts, a description of variables with their access means and influencers, and a description of environment and robot models. KOPER holds at the same time information dedicated to the planning process and to the execution phase.

We have shown how an instantiation of KOPER can be used to (1) generate a planning model for off-line computation of an initial plan, (2) monitor the plan execution by inferring the execution protocol, and (3) help in defining a new planning problem for plan repair. We have discussed a generic use case of KOPER, and illustrated our personal experience using the HiDDeN architecture with concrete experiments on multi-robot missions. In this context, KOPER also helped us in managing necessary and sufficient communication in order to ensure the best possible execution of the mission.

References

[Baclawski and Simeqi 2001] Baclawski, K., and Simeqi, A. 2001. Toward Ontology-Based Component Composition. In *OOPSLA Workshop*.

[Brugali and Scandurra 2009] Brugali, D., and Scandurra, P.

and a failure is detected

2009. Component-based robotic engineering (Part I). *IEEE Robotics Automation Magazine* 16(4).

[Chaimowicz et al. 2001] Chaimowicz, L.; Sugar, T.; Kumar, V.; and Campos, M. 2001. An architecture for tightly coupled multi-robot cooperation. *ICRA*.

[Chien et al. 2006] Chien, S.; Doyle, R.; Davies, A.; Jonsson, A.; and Lorenz, R. 2006. The Future of AI in Space. *IEEE Intelligent Systems* 21(4).

[Deplanques et al. 1996] Deplanques, P.; Yriarte, L.; Zapata, R.; and Sallantin, J. 1996. An ontology for robot modeling and testing. In *CESA Symp. on Robotics and Cybernetics*.

[Dix et al. 2003] Dix, J.; Muñoz-Avila, H.; Nau, D.; and Zhang, L. 2003. IMPACTing SHOP: putting an AI planner into a multi-agent environment. *Annals of Mathematics and AI* 37(4).

[Doherty, Kvarnström, and Heintz 2009] Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *JAAMAS* 19(3).

[Drew McDermott et al. 1998] Drew McDermott, M. G.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: the planning domain definition language. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.

[Fazil Ayan et al. 2007] Fazil Ayan, N.; Kuter, U.; Yaman, F.; and Goldman, R. 2007. HOTRiDE: hierarchical ordered task replanning in dynamic environments. In *ICAPS*.

[Gateau, Lesire, and Barbier 2010] Gateau, T.; Lesire, C.; and Barbier, M. 2010. Local plan execution and repair in a hierarchical structure of sub-teams of heterogeneous autonomous vehicles. In *ICAPS Doctoral Consortium*.

[Herzog, Jacobi, and Buchmann 2008] Herzog, A.; Jacobi, D.; and Buchmann, A. 2008. A3ME-an Agent-Based middleware approach for mixed mode environments. In *Mobile Ubiquitous Computing, Systems, Services and Technologies*.

[Kazz and Greenberg 2002] Kazz, G., and Greenberg, E. 2002. Mars Relay Operations: Application of the CCSDS Proximity-1 Space Data Link Protocol. Technical report, JPL - NASA.

[Lortal, Dhouib, and Gérard 2011] Lortal, G.; Dhouib, S.; and Gérard, S. 2011. Integrating ontological domain knowledge into a robotic DSL. In *MODELS*.

[McGann et al. 2008] McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *ICRA*.

[Merri et al. 2002] Merri, M.; Melton, B.; Valera, S.; and Parkes, A. 2002. The ECSS Packet Utilization Standard and Its Support Tool. In *SpaceOps Conference*.

[Nau et al. 2003] Nau, D.; Au, T.-C.; Ilhami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: an HTN planning system. *JAIR* 20.

[Paolucci, Shehory, and Sycara 2000] Paolucci, M.; Shehory, O.; and Sycara, K. 2000. Interleaving planning and execution in a multiagent team planning environment. *Linköping Elec. Articles in CIS* 5(18).

[Parker 1998] Parker, L. E. 1998. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Trans. on Robotics and Automation* 14.

[Schlenoff and Messina 2005] Schlenoff, C., and Messina, E. 2005. A robot ontology for urban search and rescue. In *ACM workshop on Research in knowledge representation for autonomous systems*.

[Sellami et al. 2011] Sellami, Z.; Camps, V.; Aussenac-Gilles, N.; and Rougemaille, S. 2011. Ontology Co-construction with an Adaptive Multi-Agent System: Principles and Case-Study. *Knowledge Discovery, Knowlege Engineering and Knowledge Management*.

[Sirin et al. 2004] Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*.

[Sotzing, Johnson, and Lane 2008] Sotzing, C. C.; Johnson, N.; and Lane, D. M. 2008. Improving multi-AUV coordination with hierarchical blackboard-based plan representation. In *PlanSIG*.

[Tambe 1997] Tambe, M. 1997. Towards flexible teamwork. *JAIR* 7.

[Teichteil-Königsbuch, Lesire, and Infantes 2011] Teichteil-Königsbuch, F.; Lesire, C.; and Infantes, G. 2011. A generic framework for anytime execution-driven planning in robotics. In *ICRA*.

[Tramutola and Martelli 2010] Tramutola, A., and Martelli, A. 2010. Beyond the Aurora Architecture for the new challenging applications. The Enhanced Avionics Architecture for the Exploration Missions. Technical report, Thales Alenia Space.

[Tran and Low 2008] Tran, Q., and Low, G. 2008. MOBMAS: A methodology for ontology-based multi-agent systems development. *Information and Software Technology* 50(7-8).

[Vidal et al. 2002] Vidal, R.; Shakernia, O.; Kim, H. J.; Shim, D. H.; and Sastry, S. 2002. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Trans. on Robotics and Automation* 18.

[Visentin 2007] Visentin, G. 2007. Autonomy in ESA Planetary Robotics Missions. Technical report, ESA.

[Waibel et al. 2011] Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft. 2011. RoboEarth-A World Wide Web for Robots. *IEEE Robotics Automation Magazine* 18.

[Witt et al. 2008] Witt, K. J.; Stanley, J.; Smithbauer, D.; Mandl, D.; Ly, V.; Underbrink, A.; and Metheny, M. 2008. Enabling Sensor Webs by Utilizing SWAMO for Autonomous Operations. In *NASA ESTC*.