A Hierarchical Deliberative Architecture Framework based on Goal Decomposition

Charles Lesire, Rafael Bailon-Ruiz, Magali Barbier, Christophe Grand

A Hierarchical Deliberative Architecture Framework based on Goal Decomposition

Charles Lesire[1], Rafael Bailon-Ruiz[1], Magali Barbier[1], and Christophe Grand[1]

***Abstract*— Performing a complex autonomous mission with a multi-robot system requires to integrate several deliberative approaches to perform task allocation, optimization, and execution control. Implementing such a deliberative architecture is a complex task: it requires the developer to master the decision algorithms themselves (e.g., automated planning models), to have a good knowledge of the involved robotic platforms, and to think about how these elements will be assembled as a system architecture. We propose a framework to help designing such deliberative architectures. The framework relies on the concept of a hierarchical structure of actors, each actor managing goals with specific planning or optimization approaches, and delegating sub-goals to other actors.**

I. INTRODUCTION

The use of autonomous robots (being independent robots or a multi-robot system – MRS) into real-life applications raises the challenge of designing complex deliberative capacities that may cope with different missions, the evolution of the environment, and long-term decision making. Applications like Mars exploration [1], search-and-rescue [2], naval defense [3], or service/assistance robotics [4], show that such deliberative skills must manage several mission specifications with several objectives and tasks (e.g., search, observe, report).

Hierarchical approaches have been largely used to manage such complex missions. Decomposing tasks or goals through hierarchical reasoning is indeed suited to model missions operational description. Examples of these approaches use Hierarchical Task Network (HTN) planning [5], [6], HTN planning combined with motion planning [7], [8], [9] or the use of specific procedures within a classical search approach [10]. Other works combine task or role allocation at the MRS level with some local planning algorithms [11], [12], [13]. More *reactive* approaches for decision-making also use some hierarchical decomposition of tasks or behaviors. For instance, Behavior Trees implement the autonomous behavior of a system in [14], [15].

However, using one of these approaches in a different context or mission may lead to rethink the whole deliberative architecture. In this paper, we propose an *architectural framework* for hierarchical reasoning that helps to design and manage the evolution and customization of the deliberative reasoning of an autonomous (mono- or multi-) robotic system. *Actors* [16], the main elements of this architecture, are reasoning components that manage planning and acting of specific tasks within a hierarchical goal decomposition scheme. The architecture thus integrates consistently reactive behaviors and more deliberative processes within the same decision-making architecture.

The state-of-the-art related to hierarchical reasoning and architectures is first discussed in Section II. Section III then presents the architecture designed for a MRS performing a protection mission: this application introduces the main concepts of the proposed framework. Section IV details these concepts and their implementation. Finally, Section V highlights the interest of this original framework through an estimation of the effort to develop 3 robotic use-cases: 2 mono-robot systems, plus the aforementioned MRS protection mission.

II. RELATED WORKS

Hierarchical decision-making that focuses on integrating several planning processes are often limited to the integration of two processes, one for high-level abstract decision-making, and the other for motion or trajectory planning. A planner based on the Planning Domain Definition Language (PDDL) with pre-computed trajectory patterns as basic actions is used in [10] to solve search-and-rescue missions. Partially-Observable Markov Decision Process (POMDP) policies for each individual task are called by a classical planner for high-level decision-making and task allocation in [17]. Other approaches explicitly use HTNs as decomposition models, either by integrating a specific planner to select leaf tasks of the HTN, mainly for motion planning [7], [8], [9], or combining HTN planning with other approaches, like Partial-Order Planning [18]. While these works may give some hints for the integration of several planning processes using several abstract models, they do not really provide a *framework* or methodology to integrate more diverse planning processes, and do not allow to integrate acting and reactive processes.

T-REX [19] is an architectural concept where each *reactor* manages one part of the mission by planning the corresponding system activities and controlling their execution. Reactors form a hierarchical architecture, where higher-level reactors reason on the long-term for the global mission, while lower-level reactors reason on the short-term for specialized systems. Its applications to MRS are limited to architectures where: (1) each robot has its own T-REX deliberative architecture and only exchange data with its teammates [20], [21], [22], or (2) one robot is the leader, owning the complete deliberative architecture, and sending low-level commands

*This work was not supported by any organization

[1]Authors are with ONERA/DTIS, University of Toulouse, France firstname.lastname@onera.fr

to a follower robot [23]. Furthermore, T-REX reactors are restricted to use timeline models for planning and acting.

A similar work proposed in the Mlaras architecture [24] is structured into layers, each layer corresponding to one deliberation. It hence refines the global planning problem into more concrete actions from top to bottom. The design of the architecture is essentially *declarative*, by relying intensively on PDDL to define the blocks of the system. But we claim that we also need more *operational* means to define some behaviors that cannot rely on pure automated planning. Also, their architecture contains only one deliberation per layer, which prevents dispatching goals or tasks in parallel to several sub-systems, a mandatory feature in multi-robot systems.

ROSPlan [25] allows to integrate planners based on the temporal PDDL into a ROS architecture. Its use in a two-tier MRS has been evaluated in [26]. The architecture contains only a goal allocation node at mission level and a temporal planner node at vehicles level, and no real guideline to design MRS deliberative architecture is given. The MoBaR system [27] is a three-tier architecture that combines a PDDL planner, the PLEXIL executive, and GenoM or ROS to build autonomous control architectures, but it is limited to mono-robot systems.

In conclusion, none of these works offers a real *framework* helping the design of hierarchical deliberative architectures that can moreover be used both for mono-robot systems and to decentralize the deliberation for a MRS. In this paper, our objective is to provide such a framework. Our approach combines the concept of the reasoning components *actor* (introduced in [16] and used for instance in [28]) with the concept of *goal lifecycles* [29] to supplement planning capabilities with *acting* capabilities. Controlling the execution, decomposing planned tasks into executable operations, and reacting to failures and disturbing events remain indeed critical to close the decision loop.

III. APPLICATION TO A MULTI-ROBOT PROTECTION MISSION

Let's introduce the main concepts of the deliberative architecture framework on a protection mission to be performed by a MRS. A team of ground and aerial robots have to protect an area: some robots placed at sentry positions such as buildings and crossroads detect intruders while others perform patrols surrounding the area. An example with three sentries and two patrols is given in Fig. 1. In this mission, when an intruder is detected, an identification task must be performed by some robots to identify the possible threat. Moreover, given the current situation assessment and the possible detected or identified threat, the operator can ask for the exploration of a specific area.

The MRS deliberative architecture for a team of Unmanned Ground Vehicles (UGVs) and Unmanned Aerial Vehicles (UAVs) is depicted in Fig. 2. Its components are either **actors** (rounded rectangles) or **observers** (sharp rectangles with italic labels). Deliberative functions are structured as a

Fig. 1: Specification of the *Protection* goal: the MRS must be allocated three sentry positions (depicted with yellow triangles) and two patrolling trajectories (in green and blue).

hierarchy of actors: each actor manages one kind of goal and delegates the management of sub-goals to child actors. For their part, observers analyze robot state and sensors data to generate adequate events.

The root actor, namely the *Mission Control System* (MCS), orchestrates the realization of the mission in interaction with human operators through a dedicated user interface. Decision-making in the MCS is described by simple rules that link the activation of sub-goals to either operator inputs or new detected events. Execution of these sub-goals are then delegated to the *Protection*, *Exploration* or *Identification* actors. Events triggering the execution of sub-goals in the *MCS* actor are generated by the *EnvDB* observer that manages information about the environment.

The *Protection* actor manages the execution of a protection goal by first **expanding** this goal, i.e. decomposing its specification (as illustrated in Fig. 1) into sub-goals allocated to each robot. To do so, it needs information about available robots (including their navigation skills and sensors) and the environment model (navigable paths, associated costs), which comes from the corresponding **observers**. In practice, this decomposition is computed by solving a Mixed-Integer Linear Programming (MILP) problem that minimizes the time to reach the sentry and patrol locations. The structure of the resulting decomposition is in this case a **Partial-Order Schedule** that ensures some precedence constraints between sub-goals (e.g., sentries must start before patrols). Once decomposed, the goal is **dispatched** by delegating the management of sub-goals to child actors. These actors, here directly embedded on robots, manage the execution of goals corresponding to reaching waypoints, patrolling, and performing a sentry.

Focusing on UGVs architecture, the *Patrol* actor manages a patrol goal by expanding it into a transit to reach the patrol initial position, then the patrol itself on the given trajectory, forward and backward, until a stop condition (a defined time or battery level). *Patrol* and *Transit* actors basically use some path planning on graphs to compute paths, using the robot current navigation graph as well as its state (position, battery) available in the *StateObs* observer.

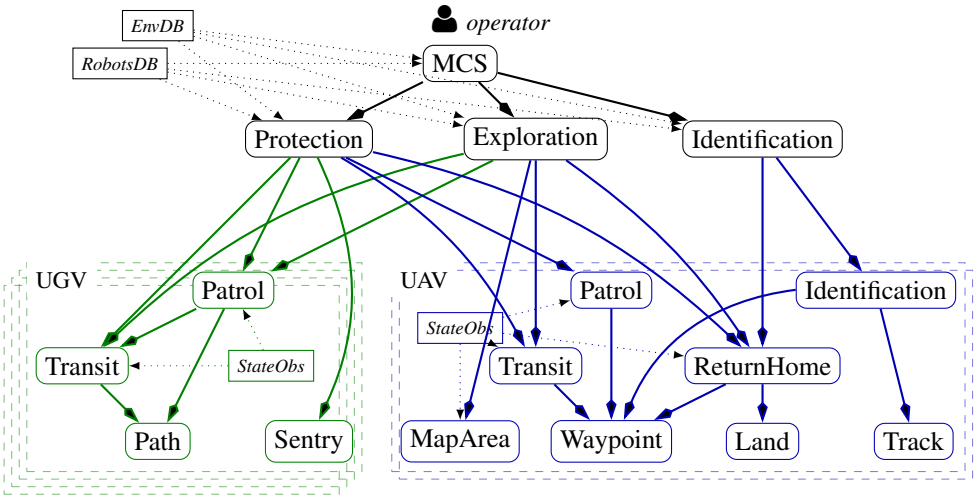


Fig. 2: MRS deliberative architecture for the protection mission. The four UGV architectures are similar, as well as the two UAV architectures, and they have then been grouped to make the figure easier to read. Plain arcs represent goal decomposition requests. Dotted arcs represent data sent from observers to actors.

Lowest-level actors that execute elementary goals through a dedicated interface with robots functional layers are called **controllers**. For instance, they execute path following or sentries by calling services or publishing to topics in a ROS-based robot architecture or using a skill-based interface [30].

Actors report on the execution of a goal to their parent actor in the hierarchy, so that the latter can **monitor** the progress of the goal being delegated. These reports can include failures, e.g., if moving on a path becomes impossible due to an environment change or a robot issue. The upper actor can then decide whether to **resolve** this goal, e.g., by computing a new path, or report again the failure to its parent actor that will for instance allocate the goal to another robot.

IV. A HIERARCHICAL DELIBERATIVE ARCHITECTURE FRAMEWORK

This section details the developed *framework* to design deliberative architectures such as the one presented for the MRS protection mission. This original framework is based on the concepts of actors hierarchically structured to support goal decomposition, and of observers managing data flow in the deliberative architecture. This framework comes with a practical implementation using the ROS2 middleware.

A. Goal Management in Actors

The proposed architectural framework relies on a hierarchy of actors that leads to a hierarchical goal decomposition. Each actor manages one type of goal and is implemented by the goal lifecycle presented in Fig. 3. This lifecyle has been originally proposed in [29], where a unique agent, the *goal reasoner* (GR), manages goals for a unique robotic system. While top goals are formulated by a user, sub-goals are managed internally, the GR inserting these sub-goals in its own queue of goals. We have leveraged this goal lifecycle concept into a hierarchical multi-actor architecture. In the description of lifecycle steps given below, A refers to the current actor that implements this lifecycle, and P refers to its parent actor that delegates to A the management of goals.

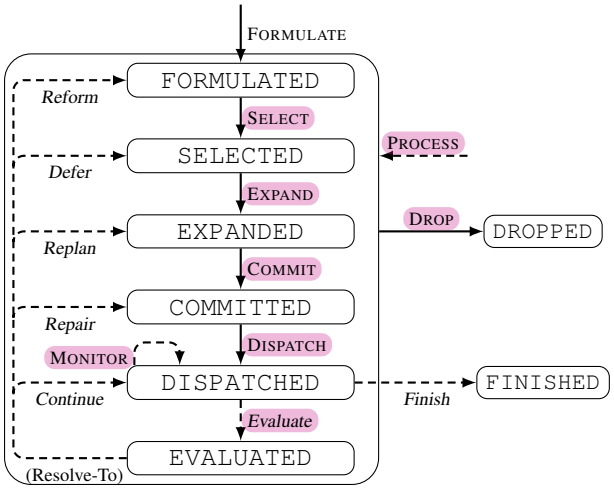


Fig. 3: Goal Lifecycle, adapted from [29]. Plain transitions are requests coming from a parent actor. Dashed transitions are internal to the actor. Transitions highlighted in pink must be implemented by the actor developer, whereas transitions with italic labels are only triggered in the developer's code.

1) Formulation: P delegates a new goal to A, through the FORMULATE request, and this goal is added to A's memory as a formulated goal.

2) Selection: P requests A to SELECT a formulated goal. It is then up to A to accept or reject this request. This is typically implemented as checking that there is no mutual exclusion with a goal already selected, or that the goal is valid in the current context (e.g., not out of bounds). For instance, the *Protection* actor of Fig. 2 verifies that enough robots are available to perform all sub-goals, while the *Transit* actors check that the target point is reachable in

the robot navigation graph.

3) Expansion: P requests A to compute some (generally one, possibly multiple) decomposition of a selected goal into sub-goals. To do so, A's developer implements the EXPAND function with the method of their choice. In the protection mission, such functions are implemented using MILP, graph theory, or hand-written parametric plans. Whatever the decomposition succeeds or fails, the result is notified to P.

4) Commitment: Given the possible decomposition returned when expanding the goal, P confirms (or chooses in case of several plans) the commitment of the goal. A must then be ready for execution, by checking the validity of the plan or asking child actors to commit sub-goals.

5) Dispatching: P requests A to actually start the goal execution. In most of the cases, it consists in A formulating its sub-goals to child actors, then going through their lifecycle until dispatching them, according to the plan structure (e.g., managing sub-goals precedence relation).

6) Monitoring: The MONITOR transition is periodically triggered by actor A while the goal is dispatched. It checks the execution status of sub-goals and reacts to reports. When the current goal is completed (i.e., all the sub-goals succeeded), the goal is finished: the `FINISHED` state is reported to P, and the goal is removed from A's memory. If any mistake or failure is reported from the sub-goal management (e.g., a sub-goal that cannot be selected or expanded), then the *Evaluate* transition is triggered.

7) Evaluation: The *Evaluate* transition is an internal transition that is triggered by the developer of A when events or reports jeopardize the goal execution. In the *Evaluate* function, the developer determines which transition to perform given the current information among the possible **resolvers** that can put back the goal upper in its lifecycle. These resolvers have the same behavior: (1) they try to put the goal back in their corresponding step, calling the corresponding transition as if it was requested by P; (2) in case of success, they trig the successive transitions to put the goal in the dispatched state again; (3) in case of failure, the resolver just above is applied, following the same behavior. For instance, resolving a goal with *Replan* will call again the expansion function, computing a new decomposition for this goal. If a decomposition is found, the goal is first committed, then dispatched. If no decomposition is found, the goal is *deferred*: it is selected, then expanded, down to dispatch. If a transition fails again, the goal is *reformed*. Reforming a goal is a specific resolver, as it does not try to select the goal again, but reports to P that the goal has been reformed. It is then up to P to decide if the goal must be dropped, or if it must be tried again.

The behavior of resolvers is automatically managed in the framework implementation: the developer has just to call the desired resolver in its *Evaluate* implementation, and lifecycle management and reports toward P are automatically performed according to the success or failure of transitions.

8) Dropping: The DROP transition is tailored by actor A's developer to remove the goal from memory and to drop sub-goals that have already been delegated to other actors, when it is required by actor P.

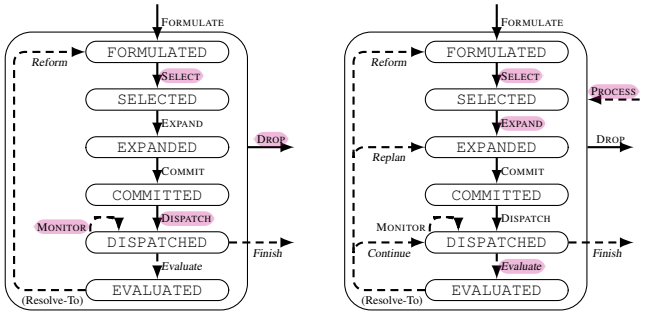
9) Processing events: Actors can subscribe to **events** (described later). The arrival of new events is managed in the PROCESS transition. According to current situation and state of goals, A's developer decides to call the *Evaluate* transition of some goals to determine how to react to events.

In the protection mission (Fig. 2), when receiving an event about the detection of a new intruder, *MCS* replans the current mission to include an identification sub-goal in addition to the existing sub-goals (protection and/or exploration).

B. Actor Patterns

In most cases, a full goal lifecycle is not necessary. We defined then two specific lifecycle patterns in the framework: controllers, and plan/replan behaviors based on Partial-Order Schedules (POS).

Controllers are the actors at the lowest level of the deliberative architecture. They cannot create sub-goals and delegate them to other actors, but instead manage one goal execution with a direct interface to the robot functional layer. Therefore, in their lifecycle shown in Fig. 4a, the EXPAND and COMMIT transitions do nothing specific, as no decomposition must be computed. Also, the controller cannot process external events but only reacts to execution reports from the functional layer, handled in the MONITOR transition. Only reactions are then to *Finish* the goal in case of success, or to *Reform* it in case of failure.



(a) Controller (b) POS and TimedPOS

Fig. 4: Specific lifecycles of some actor patterns (same legend as in Fig. 3).

POS actors manage goal decomposition where sub-goals have precedence relations. Their corresponding lifecycle is depicted in Fig. 4b: due to the known structure of the plan, the COMMIT, DISPATCH, MONITOR and DROP transitions are already implemented: a sub-goal can be executed only once all its predecessors have finished (i.e., successfully). When an error is reported on a sub-goal (which has then been reformed), the evaluation may lead to either consider the sub-goal is finished and *continue* the plan execution, *replan* to find a new plan, or *reform* the goal itself and report to the parent actor.

Timed POS actors extend POS actors by considering that sub-goals have an optional dispatch time. They behave as POS actors regarding the precedence checks of sub-goals, but instead of dispatching immediately, the actors set up a

timer that delays the execution of sub-goals. These actors are useful for plans that contain time-constrained actions. They can also handle hybrid plans with timed and not timed sub-goals.

C. Data Management

The hierarchy of actors manages goal decomposition, including planning (in the expansion step) and execution control (when dispatching and then monitoring sub-goals). Goals and their current state in their respective lifecycles are the only data exchanged between actors in this hierarchy.

However, actors have to manage other information to reason and to take decisions. Components that provide data on-demand to the actors are called **observers**.

In the protection mission architecture (Fig. 2), *StateObs* observers at the robot level manage data related to the robot internal state: current position, battery level, current map, state of sensors, etc. These data are gathered through a specific interface, typically by subscribing to ROS topics. Observers can for instance provide data that actually aggregate information coming from several ROS topics. From the point of view of the design process, having these data centralized into observers has the advantage to make actors independent on *how* data must be gathered: actors are not bound to specific ROS topics but instead use the data management interface provided by the framework. This makes the actors easier to maintain and reusable between architectures, following the separation of concerns paradigm [31].

For their part, *EnvDB* and *RobotDB* observers manage static data that describe the environment in which the mission takes place (e.g., known and unknown areas, reachable zones) and the set of available robots along with their features (e.g., equipped with sensors needed for sentries, nominal speed or battery consumption rate). These information are used by the multi-robot actors that are implementing the task allocation algorithms.

D. Event Management

In some situations, actors do not want to demand the data values, but instead want to be informed when these values change or satisfy some given properties. In our framework, this is formalized using the concept of **events**. Events are specified using the Metric Temporal Logic (MTL) evaluated on data values [32]. This logic allows to have a clear specification of what these events represent, and to use standard tools for their runtime evaluation [33].

Events are then defined using MTL within observers, based on the data they manage. When actors subscribe to events, they are notified each time the event MTL formula becomes true, through the actors' PROCESS transition. It is then up to the actor's developer to decide how to react to these events. Some default behaviors are already implemented in the framework and can be used by the developer, such as systematically replanning the dispatched goals.

In the protection mission architecture (Fig. 2), the intruder detection event is defined using the MTL formula $H_5(intruder \neq \emptyset)$, specifying that the value of data $intruder$ has not been empty for the last 5 time steps (using the *historically* operator).

E. Framework Implementation

The framework has been implemented in Python language using ROS2 middleware. Each component, actor or observer, is then a ROS2 node. Communications between these components (i.e., goal requests, getting data, events) are available through a high-level interface: the component developer has never to manipulate ROS primitives to make these connections. The use of bare ROS functions is only needed when developing a controller that must interface with the functional layer through ROS, or when observers need to gather data values in the functional layer from ROS.

Framework implementation only relies on ROS2 topics, with a fine tuning of the underlying Data Distribution Service (DDS) quality of service: it allows to setup the level of reliability depending on the requirements of each connection, and makes use of the liveliness property to detect (dis)connections between actors. The framework library is available at:

`http://oara-architecture.gitlab.io/`

V. DESIGN OF DELIBERATIVE ARCHITECTURES

The framework has been applied to 4 use-cases: (1) the *travel* standard problem in HTN planning used as a tutorial in the library documentation, (2) the MRS protection mission of Section III, (3) an Earth Observing Satellite (EOS) mission controller that manages the dynamic acquisition of the best targets to photograph, and (4) an Autonomous Underwater Vehicle (AUV) surveying an area.

Use-cases (3) and (4) are mono-robot architectures in environments where a high level of autonomy is required, while use-case (2) is a MRS with the need to decentralize the decision between multi-robot components and individual tasks. In this latter case, field experiments with 4 UGVs and 2 UAVs implemented the architecture illustrated in Fig. 2. During these experiments, 44 ROS2 nodes (excluding robots functional layers nodes) were deployed and distributed over 7 computing units.

We assess the relevance of the proposed framework by estimating the development effort of its library as well as of use-cases instances. To do so, we counted the number of Lines of Code (LoC) of ROS packages and estimated the effort in person-month (p.m) using the COCOMO approach [34]. Results are reported in Tables I and II.

Package	LoC	Estimated effort (p.m)
`oara`	3,404	15.66*
`oara_interfaces`	162	0.35†
`oara_common_interfaces`	432	0.99†
Total	3,998	17.01

TABLE I: LoC and estimated development effort of framework packages. The `oara` package contains the complete library to develop actors and observers, while other packages contain ROS message and service types.

Use-case	Components	LoC	Estimated effort (p.m)
Travel	5	170	$0.37^{\dagger}$
MRS	20	1,853	$4.59^{\dagger}$
EOS	11	1,235	$2.99^{\dagger}$
AUV	5	307	$0.69^{\dagger}$

TABLE II: Number of actor and observer components, LoC and estimated development effort of use-cases.

Even if the estimated effort is not to be taken as an absolute indication, it allows to evaluate the benefit of using the proposed framework. The development of each use-case required little effort regarding to quite complex architectures. Given the estimated effort of the framework itself, we can guess that the effort of developing each use-case architecture without any framework would have been several times higher.

VI. CONCLUSION

This paper describes a framework to design deliberative architectures for robotic systems. This framework relies on the concept of a hierarchical structure of actors. Each actor manages one type of goal, through a formal *goal lifecycle*, in which the actor developer can implement specific steps. Sub-goals are then delegated to child actors in the hierarchical architecture. In addition to actors, the framework uses observers to store data and emit events.

This framework has been applied to several use-cases, and the identified advantages are twofold: first, the framework concepts help to design effective deliberative architectures, and second, it reduces the development effort, as shown by measuring the lines of codes of each application.

REFERENCES

[1] A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael, "A Goal-Oriented Autonomous Controller for space exploration," in *ASTRA*, Noordwijk, the Netherlands, 2011.

[2] H. Kitano and S. Tadokoro, "RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems," *AI Magazine*, vol. 22, no. 1, 2001.

[3] E. Chauveau, C. Lesire, and F. Chaillan, "Integration of Autonomous Heterogeneous Systems for Decision Making Autonomy in Naval Defense: a Position Paper," in *OCEANS*, Marseille, France, 2019.

[4] M. Di Rocco, F. Pecora, and A. Saffiotti, "When robots are late: Configuration planning for multiple robots with dynamic goals," in *IROS*, Tokyo, Japan, 2013.

[5] K. Erol, J. A. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *AAAI*, Seattle, WA, USA, 1994.

[6] C. Lesire, G. Infantes, T. Gateau, and M. Barbier, "A distributed architecture for supervision of autonomous multi-robot missions," *Autonomous Robots*, vol. 40, no. 7, pp. 1343–1362, 2016.

[7] L. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, Shangai, China, 2011.

[8] L. de Silva, A. K. Pandey, and R. Alami, "An interface for interleaved symbolic-geometric planning and backtracking," in *IROS*, Tokyo, Japan, 2013.

[9] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229 – 265, 2017.

*estimated using the COCOMO basic Embedded model, which corresponds to complex systems, as the `oara` library has required a deep understanding of the ROS2 and DDS primitives.

†estimated using the COCOMO basic Organic model, which corresponds to small and simple software units.

[10] S. Bernardini, M. Fox, and D. Long, "Combining temporal planning with probabilistic reasoning for autonomous surveillance missions," *Autonomous Robots*, vol. 41, no. 1, pp. 181–203, 2017.

[11] R. Zlot and A. Stentz, "Market-Based Multirobot Coordination Using Task Abstraction," in *FSR*, Lake Yamanaka, Japan, 2003.

[12] L. Hunsberger and B. Grosz, "A combinatorial auction for collaborative planning," in *ICMAS*, Boston, MA, USA, 2000.

[13] Y. Carreno, E. Pairet, Y. Petillot, and R. P. A. Petrick, "A Decentralised Strategy for Heterogeneous AUV Missions via Goal Distribution and Temporal Planning," in *ICAPS*, Nancy, France, 2020.

[14] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE T-RO*, vol. 33, no. 2, pp. 372–389, 2017.

[15] A. Albore, D. Doose, C. Grand, C. Lesire, and A. Manecy, "Skill-Based Architecture Development for Online Mission Reconfiguration and Failure Management," in *RoSE@ICSE*, Madrid, Spain, 2021.

[16] M. Ghallab, D. S. Nau, and P. Traverso, "The actor's view of automated planning and acting: A position paper," *Artificial Intelligence*, vol. 208, pp. 1–17, 2014.

[17] M. Hanheide, M. Göbelbecker, G. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, H. Zender, G.-J. Kruijff, N. Hawes, and J. Wyatt, "Robot task planning and explanation in open and uncertain worlds," *Artificial Intelligence*, vol. 247, pp. 119 – 150, 2017.

[18] P. Bechon, C. Lesire, and M. Barbier, "Hybrid planning and distributed iterative repair for multi-robot missions with communication losses," *Autonomous Robots*, vol. 44, no. 3-4, pp. 505–531, 2020.

[19] 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.

[20] A. S. Ferreira, M. Costa, F. Py, J. Pinto, M. A. Silva, A. Nimmo-Smith, T. A. Johansen, J. B. de Sousa, and K. Rajan, "Advancing multi-vehicle deployments in oceanographic field experiments," *Autonomous Robots*, vol. 43, no. 6, 2019.

[21] J. Das, F. Py, T. Maughan, T. O'Reilly, M. Messié, J. Ryan, G. Sukhatme, and K. Rajan, "Coordinated sampling of dynamic oceanographic features with underwater vehicles and drifters," *IJRR*, vol. 31, no. 5, pp. 626–646, 2012.

[22] A. Belbachir, F. Ingrand, and S. Lacroix, "A cooperative architecture for target localization using multiple AUVs," *Intelligent Service Robotics*, vol. 5, no. 2, pp. 119–132, 2012.

[23] F. Ropero, P. Muñoz, and M. D. R-Moreno, "ARIES: An Autonomous Controller For Multirobot Cooperation," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 3, pp. 40–55, 2019.

[24] J. C. González, A. García-Olaya, and F. Fernández, "Multi-Layered Multi-Robot Control Architecture for the Robocup Logistics League," in *ICARSC*, Azores, Portugal, 2020.

[25] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras, "Rosplan: Planning in the robot operating system," in *ICAPS*, Jerusalem, Israel, 2015.

[26] Y. Carreno, R. P. A. Petrick, and Y. Petillot, "Multi-agent Strategy for Marine Applications via Temporal Planning," in *Int. Conf. on AI and Knowledge Engineering (AIKE)*, Cagliari, Italy, 2019.

[27] P. Muñoz, M. Moreno, D. Barrero, and F. Ropero, "MOBAR: a Hierarchical Action-Oriented Autonomous Control Architecture," *Journal of Intelligent and Robotic Systems*, vol. 94, pp. 745–760, 2019.

[28] S. Patra, M. Ghallab, D. S. Nau, and P. Traverso, "Acting and Planning Using Operational Models," in *AAAI*, Honolulu, HI, USA, 2019.

[29] M. Roberts, V. Shivashankar, R. Alford, M. Leece, S. Gupta, and D. W. Aha, "Goal Reasoning, Planning, and Acting with ActorSim, The Actor Simulator," in *Annual Conference on Advances in Cognitive Systems*, Evanston, IL, USA, 2016.

[30] C. Lesire, D. Doose, and C. Grand, "Formalization of Robot Skills with Descriptive and Operational Models," in *IROS*, Las Vegas, NV, USA, 2020.

[31] M. Radestock and S. Eisenbach, "Coordination in evolving systems," *Trends in Distributed Systems, CORBA and Beyond*, 1996.

[32] J. Ouaknine and J. Worrell, "On the decidability of metric temporal logic," in *IEEE Symposium on Logic in Computer Science*, Chicago, IL, USA, 2005.

[33] D. Ulus, "Online monitoring of metric temporal logic using sequential networks," *CoRR*, vol. abs/1901.00175, 2019.

[34] B. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981.