Plan Verbalisation for Robots Acting in Dynamic Environments

Konstantinos Gavriilidis [1,2*], Yaniel Carreno [1,2*], Andrea Munafo [3], Wei Pang [1,2], Ronald P. A. Petrick [1,2], Helen Hastie [1,2]

[1] Edinburgh Centre for Robotics, Edinburgh, UK
[2] Department of Computer Science, Heriot-Watt University, Edinburgh, UK
[3] Seebyte Ltd, 30 Queensferry Rd, Edinburgh, UK
{kg47, Y.Carreno, W.Pang, R.Petrick, H.Hastie}@hw.ac.uk
andrea.munafo@seebyte.com

Abstract

Automated planning provides the tools for intelligent behaviours in robotic platforms deployed in real-world environments. The complexity of these domains requires planning models that support the system's dynamics. This results in AI planning approaches often generating plans where the reasoning around the solution remains obscure for the operator/user. This lack of transparency can reduce trust, results in frequent interventions, and ultimately represents a barrier to adopting autonomous systems. Explanations of behaviour in an easy-to-understand manner, such as in natural language, can help the user comprehend the reasoning behind autonomous actions and help build an accurate mental model. This paper presents an approach for a type of explanation, namely plan verbalisation, that considers the properties of the planning model and describes the system behaviour during plan execution, including replanning and plan repair. We use natural language techniques to support the disambiguation of the robot decision-making process, considering the planning model encapsulated using the Planning Domain Definition Language (PDDL). The system is evaluated using an Autonomous Underwater Vehicle (AUV) inspection use case.

1 Motivation and Introduction

Robotic platforms are frequently deployed to carry out complex tasks such as exploration, maintenance, and manufacturing to improve mission quality in hazardous conditions. Critical applications in different areas such as nuclear (cf. `rainhub.org.uk`), offshore energy (cf. `orcahub.org`), and renewables (cf. `mimreesystem.co.uk`) can enter extreme environments and undertake high-risk missions where an operator can supervise the activity from a safe distance. Automated planning generates planning solutions that achieve tasks with different levels of complexity and requirements (e.g., heterogeneous robots, collaborative actions, coordinated actions, etc.), minimising their costs. Planning solvers can generate solutions in short periods, and, if necessary, planners can re-adjust the behaviour of systems in case of unexpected circumstances.

However, with complex plans and highly dynamic environments, it is not always clear how a system may act in a given situation or why it behaves in a certain way. This lack of transparency is a recognised problem going forward (cf. the forthcoming IEEE P7001 standard on transparency) and can result in a lack of confidence, trust and ultimately prevent adoption. Furthermore, as systems are adopted into the field, there is a need for an audit trail of actions and the planner decisions regarding the action sequence.

Currently, there is a growing interest in explainable planning (Chakraborti, Sreedharan, and Kambhampati 2020), where agents can explain their decisions to operators. This approach can increase user trust towards robots and find the right balance between human intervention and autonomy. However, instead of directly making the agents explainable, adding a wrapper application that represents concepts such as planning either using visualisation or natural language (NL) is possible. Our interest lies in natural language explanations and their ability to personalise outputs depending on the preferred amount of knowledge and expertise of the user (Garcia et al. 2018). We are interested in verbalising the planning aspects of robots to help users understand discrepancies in terms of expected and actual robot behaviour.

This paper contributes to explainable planning with natural language explanations with the provision of the following: (i) introduction of an Autonomous Underwater Vehicle (AUV) domain; (ii) planning explanation considering verbalisation of the action progress during plan execution; and (iii) clarification of action failure and replanning using natural language. Our approach provides tools to verbalise missions initial state, goal state, plan solution and unexpected outcomes, including replanning or plan repair.

The rest of this paper is organised as follows. We begin by discussing relevant work that relates automated planning with natural language. We then briefly describe the use case for our implementation and provide technical details of our approach to plan verbalisation and the planning system. Finally, we provide an example of our implementation and conclude with the findings of our work and future directions.

2 Related Work

Natural language has long been recognised as an effective medium to convey information about planning aspects, and objective completion to users (Sohrabi, Baier, and McIlraith 2011; Fox, Long, and Magazzeni 2017; Robb et al. 2018).

*Authors have contributed equally in this work.

It can connect the dots between mission data and display content in an informative manner to reduce uncertainty. To retrieve that information, explanation interfaces that relate planning data to meaningful content are needed to elaborate on the processes and decisions behind a choice of plan (Fox, Long, and Magazzeni 2017). We retrieve data during action execution and plan failures to generate explanations about mission outcomes with our approach.

Other works that focus on the personalisation of explainable planning (Chakraborti, Sreedharan, and Kambhampati 2020) outline the users that would potentially be interested in planning or decision-making explanations. Authors describe the implemented parts of autonomous agents that require explainability (e.g., plan or policy clarifications). The critical point of this work is that a certain level of personalisation in natural language outputs is necessary to prevent the mental overload of users or loss of interest. To refine the user experience in our implementation, we limit the number of technical details in outputs and facilitate mission inspections by users with low planning expertise.

Prior work on plan verbalisation has also focused on knowledge acquisition and explanation generation (Sridharan and Meadows 2019), or the effect of explanations on user trust during human-robot teaming (Hastie, Liu, and Patron 2017; Nikolaidis et al. 2018). We focus on action explanations to provide clarity, and we only offer unilateral answers from the agent. Furthermore, there have been many applications that verbalise different types of content, such as robot localisation (Rosenthal, Selvaraj, and Veloso 2016; Moon, Magazzeni, and Cashmore 2019), sensor data (Hastie, Liu, and Patron 2016), or action failures (Thielstrom et al. 2020). Compared to these approaches, we aim to verify action completion when cancellation occurs. We present the error and the new set of actions to be executed with replanning.

Explanation models can also be used as wrapper services to facilitate user awareness without disrupting the design of autonomous systems (Adadi and Berrada 2018). A model-agnostic application that follows this technique uses an interface that contains a set of constrained questions, which leads to planning indications as a form of answer (Cashmore et al. 2019). Our model also follows this example and is applied to the planning system as an explanation layer.

A key tool in our approach is the use of ontologies, which have a long history in planning (McCluskey and Cresswell 2005; Bouillet et al. 2007; Cioffi and Thompson 2007). Previous work on knowledge representation has examined the use of sensor data from plan execution along with a knowledge base to relate that data to symbolic concepts (Tenorth and Beetz 2009). Additional work fixes differences between a planning domain and an ontology (McNeill and Bundy 2007) or extends the functionality of a planner by introducing new information that leads to the formulation of new goals (Babli, Onaindia, and Marzal 2019). Ontologies can also be utilised to create a Natural Language Generation system that derives content from them (Galanis and Androutsopoulos 2007). Literature shows approaches that either utilised abstract ontologies with concept and axiom definitions (TBox) or knowledge bases that combine these concepts/axioms with instances and concept/role assertions

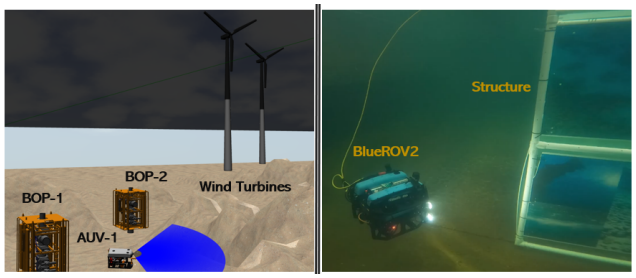


Figure 1: Illustration of the simulation (left) and real (right) domain used to implement AUV exploration missions.

(ABox) (Miguelanez et al. 2010). Our work combines planning with TBox ontologies to efficiently describe the current state of a robot and to indicate whether it is capable of performing an action.

3 Domain and Problem Definition

As a running example throughout the paper, we consider an AUV with electrical manipulators, stereo cameras and sonar, which is used to implement multiple tasks in the underwater domain such as: (i) seabed mapping, (ii) structure reconstruction, (iii) inspection of valves and modification of their handles, and (iv) rock inspection to assess the soil quality for environmental purposes. Here, we present an updated version of the underwater domain used in (Carreno et al. 2020). Figure 1 shows the simulated (left) and real (right) scenario associated to our domain `auv_inspection`. The Simulation Scenario and the Real Scenario share similar properties. Therefore, we can assume we have a single model that responds to the `auv_inspection` domain. We adopt the Planning Domain Definition Language (PDDL) with temporal notions (Fox and Long 2003) to describe our domain and problem[1].

Domain Description (Simulation Scenario): An offshore scenario (Figure 1, left) includes a set of blowout preventers (BOPs), structures with a valve attached that can be open or closed (the valve state is known at planning time). In addition, the environment presents multiple wind turbines, which require regular inspection of their bases. The structure's coordinates are known, and the AUV does not have any initial knowledge about the seabed characteristics. This scenario allows the implementation of missions that cover all types of tasks described in the `auv_inspection` domain.

Domain Description (Real Scenario): A real underwater scenario (Figure 1, right) with two structures was built at Heriot-Watt University. The AUV knows the structures' positions but lacks knowledge about the remaining features that define the environment (e.g., a structure's shapes, floor irregularities, obstacles, etc.). This scenario is restricted to missions associated with structure mapping and exploration.

Our domain contains eight temporal actions. The `navigation` action is used for the AUV to navigate the

[1]In https://github.com/konsgavriil/keps-auv-inspection.git, we present the full domain and problem.

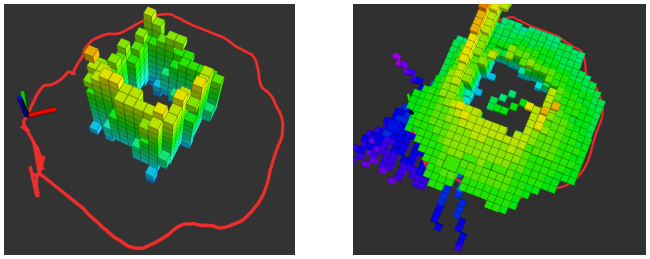
environment, taking as a reference an initial (?wpi) and final (?wpf) point. Implementing this action allows the inspection of the environment and the acquisition of data regarding unknown features. The map action is used to implement point-to-point structure mapping using specialised sensors and algorithms. The manipulation action manipulates the valve using an actuator, in our case, a robotics arm. The rock-inspection action is a sensing action that evaluates the soil quality by analysing rocks. Other domain durative actions that share the same parameters (?r - robot and ?wp - point) are communicate, recharge, hardware-repair, and recover. The communicate action is required to notify the base about plan implementation updates and failure. The recharge action plays a fundamental role in maintaining the robot under optimal energy levels to execute all mission goals. The hardware-repair action is required to implement regular checks in the hardware to avoid total failures. The action is executed when the AUV's battery goes under a particular threshold, defined as attending to the AUV's characteristics. The communicate, the recharge, and the hardware-repair actions require the AUV to be positioned at a point that holds the predicate (surfpoint_at ?r - robot ?wp - point). Finally, the recover action makes the robot return to a safe position (defined by the operator or domain designer) at the end of the mission.

In our domain, we define a set of properties associated with the AUV that affect the actions the robot can implement at each time step. For instance, the domain predicates define a set of properties associated with the robot's capabilities such as slam_equipped, rock_analyser, and arm_equipped. The slam_equipped property allows the system to implement the mapping of a structure. This predicate represents a precondition to implement the map action. The rock_analyser property indicates that the AUV can make an evaluation of a rock. The arm_equipped property is a precondition for implementing the manipulation action as the robot requires this actuator to turn the valve. Another set of AUV properties and environment features are introduced as :functions. For the environment, an example is the (distance ?wpi ?wpf - point) function, which describes the distance between two different points. Instances of this function introduce knowledge regarding the position of all mission points concerning the AUV's location at any time. For the AUV, examples enclose properties such as the battery_level, current energy and robot's speed.

Our problem defines a robot called auv and a set of points (e.g., wp0, wp1, etc.) that describe points of interest in the environment. The problem's initial state (:init) defines the auv capabilities by introducing instances that describe the AUV's sensory system and actuators; the connected points in the environment (e.g., (connected wp10 wp11)), which represent a precondition to implement the navigation between points; AUV battery properties such as its maximum, minimum and current voltage level (e.g., (= (battery_level auv) 15.6)); the distance between points (e.g., (= (distance wp10 wp11) 0.6)); amongst other properties. The problem also defines the mission goals. The Simulation Scenario allows

Time:	(Action Name)	[Duration]
0.000:	(navigation auv wp0 wp10)	[10.000]
10.002:	(map auv slam wp10 wp11)	[7.500]
17.503:	(map auv slam wp11 wp12)	[7.500]
25.004:	(map auv slam wp12 wp13)	[7.500]
32.505:	(map auv slam wp13 wp14)	[7.500]
40.006:	(map auv slam wp14 wp15)	[7.500]
47.507:	(map auv slam wp15 wp16)	[7.500]
55.008:	(map auv slam wp16 wp10)	[7.500]
62.509:	(navigation auv wp10 wp0)	[10.000]
72.510:	(navigation auv wp0 wp27)	[5.000]
77.511:	(navigation auv wp27 wp26)	[3.000]
80.512:	(rock-inspection auv wp26 rs)	[4.000]
84.513:	(navigation auv wp26 wp25)	[3.000]
87.514:	(manipulation auv wp25 arm)	[5.000]
92.515:	(navigation auv wp25 wp20)	[2.000]
94.516:	(recover auv wp20)	[1.000]

Figure 2: Temporal plan solution for the inspection domain.



(a) (b)

Figure 3: **(a)** shows the map of a BOP by implementing the explored goals in the plan; **(b)** the seabed map around the BOP structure after implementing the exploration goals.

the implementation of all goals (e.g., point_mapped wp10, recovered wp20, etc.) defined in the problem. However, for the real scenario, goals (valve_turned wp25) and (rock_collected wp26) are removed. Instances of goal (point_mapped ?wp - point) describe a circular path around a structure that requires mapping.

Figure 2 shows an optimal plan solution for the example domain and problem (obtained with the OPTIC (Benton, Coles, and Coles 2012) planner). The implementation of the navigation actions in the plan allows the inspection and mapping of a structure while acquiring other features of the underwater environment in the area. Figure 3 shows the real implementation of the plan actions (actions 2-8) regarding mapping the structure. Upon closer examination, the plan solution does not appear straightforward to understand for non-planning experts. For instance, the reason the AUV implements a set of actions with the same duration in a particular sequence can be challenging to comprehend for planner users. The reasoning behind the planning solution (using the format in Figure 2) requires users to understand PDDL semantics and the domain specifics. Therefore, additional tools are required to present planning solutions in a more friendly manner that allow users to have more detailed information about the AUV behaviours.

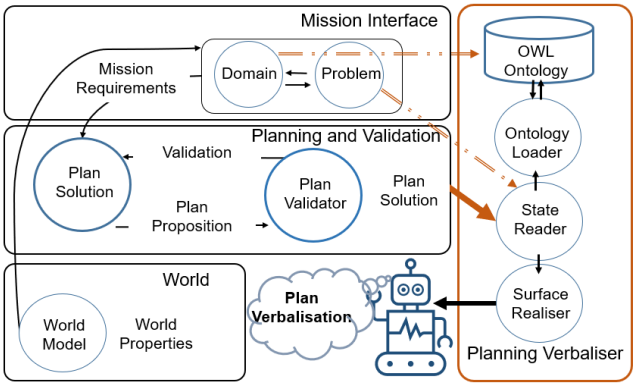


Figure 4: Pipeline architecture for plan verbalisation. The Planning Verbaliser shows the workflow for NL generation.

4 System Overview: Plan Verbaliser

Our system has the primary goal of verbalising planning structures and decisions during plan execution in dynamic environments. This includes all key elements of the planning cycle: plan generation, execution, monitoring and replanning. Given a description of the problem and the initial plan, we summarise plan execution by presenting the initial and goal states and the action steps in natural language. We track actions during execution and check whether they have been achieved or cancelled due to errors. In case of errors, we check which action has failed, and justify the cause by monitoring the process in charge of identifying mission failures. We use an underwater domain to highlight the functionality of our approach. However, both the ontology and verbaliser are scalable to other applications as a goal is to build a domain-independent solution.

The problem and initial plan explanation system is comprised of the following parts:

State Reader parses a PDDL problem, a plan and validation information about the mission and actions.

OWL Ontology is an OWL representation (McGuinness, Van Harmelen et al. 2004) of the planning domain, containing the same domain logic by replacing PDDL elements (types, actions and predicates/functions) with OWL items (entities, object properties and data properties). Upon any state updates, a reasoner can be used to infer facts about mission outcomes and derive more content for our explanations.

Ontology Loader loads the OWL ontology and interacts with the StateReader. Once a problem is retrieved and all mission elements are defined, the loader creates new entities according to the initial state. Then, after each action step, it updates all entity states of the ontology.

Surface Realiser contains a set of methods for generating the final verbalisations. The methods include the SimpleNLG package (Gatt and Reiter 2009) to apply syntactic corrections and tense modifications.

The usual workflow (see Figure 4) is that the validation step retrieves the mission interface artefacts, checks if the plan solves the problem and produces the predicate states per action step. Next, the StateReader retrieves the PDDL documents and parses them to detect entities and their states during the initial state, the goal state and after each plan step. Concurrently, an OWL ontology we have manually created is loaded with the OWLReady2 package (Lamy 2017) in the OntologyLoader module. As a result, the entities and their states are initialised in the ontology as individuals and upon any change that occurs after each action, the ontology is also updated. Currently, the ontology is not used to derive content for the verbalisation, since it could not directly access the predicates during plan execution; we are addressing this problem as future work. Finally, the static content provided by the StateReader is verbalised using the SurfaceRealiser. The initial plan verbalisation contains information regarding the world initial state (e.g., `auv` *is equipped with a* `sonar` *sensor*, `auv` *manipulates a valve at* `wp25`., etc.).

Focusing on a specific action from the domain presented in Section 3, we now demonstrate how our system works by describing the steps for explaining a `navigation` action. As an example, we have a robot called `auv`, which is at an initial location `wp0` and is available to undertake any task. The StateReader extracts the initial state of `auv`, strips it of any characters that are not needed and keeps only the identifier and the parameters (i.e. `at auv wp0`; `available auv`). Before the state is verbalised, the OntologyLoader retrieves this information and updates the ontology. In order to verbalise each action, our system confirms that the robot is able to perform the task with the use of *Class Constraints* that we have defined in the ontology. For example, a robot is capable of performing a navigation action only if it inherits the *NavigationRobot* class. To allocate this property to a robot, we run the HermiT reasoner, considering its capability to reclassify individuals based on allocated object properties (Glimm et al. 2014), and find out if that is the case. The reasoner will classify `auv` as a NavigationRobot only if it is located at the correct location and it is available. Using this methodology, we can include the action preconditions and effects during plan verbalisation by representing these changes in the ontology.

Action Explanations Once the user has been briefed about the problem and plan solution, plan execution commences and action explanations are generated. Our interest in action implementation focuses on three elements: (i) action initialisation, (ii) action feedback, and (iii) action updates. An example of the inputs and their verbalisation for action explanations are shown in Table 1. From the action initialisation, we obtain information regarding action parameters, starting time and its duration. The action feedback provides information about action completion (e.g., action achieved, action cancelled, etc.). The action id is used to identify the parameters and name of the action of interest. The action updates provide data associated with the most recent action, the System Framework embedded in our robot, called for execution. For instance, in Table 1, the Action Interface (*A-Int*) for navigation, *Navigation A-Int*, provides knowledge regarding the interface connecting with robot controllers and low-level algorithms to solve the action in the plan. We accumulate

`0.000: navigation (auv wp0 wp10) [10.000]`
Action Initialisation: **Components To Extract**: *action id*: `1` *name*: `navigation` *parameters*: `(auv wp0 wp10)` *dispatch time*: `0.000` *duration*: `10.000` **Verbalisation**: `auv` *has started moving from* `wp0` *to* `wp10`. *The action started at* `0.000` *sec. with duration* `10.000` *sec.*
Action Feedback: **Components to Extract**: *action id*: `1` *action status*: i.e., `enabled` or `achieved` **Verbalisation**: `Auv` *is now located at* `wp10`.
Action Updates: **Components to Extract**: *action interface*: i.e., ***Navigation A-Int.***, ***Map A-Int.***, etc. **Verbalisation**: *Latest update received from* ***Navigation A-Int.***

Table 1: Verbalisation for `navigation` action.

this information and verbalise it using the Natural Language Generation (NLG) system mentioned in this section.

Explanations for Plan Failure and Replanning Verbalisation of plan failures and replanning requires structured data from the system responsible for failure detection and planning state evaluation (see Section 5). Specifically, we retrieve a *notification type* describing the nature of the error (e.g., hardware, battery, localisation, etc.), a *risk level* that informs how much the error compromises the mission (e.g., high, standard, and low), and *specifics* that provide information regarding error types. For instance, the specifics for the notification type hardware can detail the part of the hardware affected or the notification reason (e.g., thrusters, sensors, water leak, etc.). The Planning Verbaliser retrieves these messages to express plan failure and mission replanning/repair characteristics using NL. The predefined structure of error outputs facilitated a template-based approach that can be used across different domains.

5 System Overview: Mission Planning

This section provides an overview of the main elements of the Planning System, which integrates with the Planning Verbaliser to offer a verbal explanation of plan solutions during plan execution. This system allows the description of plan implementation in a dynamic environment considering the plan generation, execution, monitoring, replanning cycle. Therefore, users can have updated information regarding action execution, action implementation failures, replanning, and variations in the mission goal set.

Our Goal-Based Mission Planning System (Carreno et al. 2021) integrates the Situational Evaluation and Awareness (SEA) and Online Planning components. This system combines planning, knowledge representation and decision making to achieve high-level mission goals, while maintaining mission survivability and improving robustness. SEA acts as a bridge between high-level planning and low-level mission execution systems. This component supports a dynamic evaluation of the state to provide a goal completion assessment for local recovery and global missions using the Online Planning system. The Mission Planning framework offers a connection with low-level algorithms that support the SEA decision-making process when it evaluates possible failures and proposes alternative solutions to the planning system to deal with the dynamics of the environment. The Mission Planning framework is vital because it enables users to have updated knowledge regarding plan execution, including notifications about mission failure (causes) while keeping a record of the completed and uncompleted goals. In addition, the alternative solutions proposed by the SEA component to overcome failures enhance plan survivability and quality. SEA encapsulates a library with failure types, which extends the system ontology. The library is created using past experience regarding anomaly and fault detection for the different domains (e.g., underwater, aerial, and terrestrial).

Algorithm 1: Mission Planning

Input: $\mathcal{MR}$: Global Mission Requirements.
Input: $\mathcal{IMR}$: Intermediate Mission Requirements.
Input: γ: Current Knowledge.
Output: $\mathcal{MK}$: Mission Knowledge.
Output: $\mathcal{FU}$: Failure Update.

```
1  begin
2      i ← ExtractGlobalMGoals(MR)
3      IMR ← MR
4      while not i ← ∅ do
5          P ← GenerateProblem(IMR)
6          Π_t ← GeneratePlan(P)
7          F_{i−1}.CancelCurrentPlan()
8          F_i.DispatchPlan(Π_t)
9          p.CheckSEA(IMR, γ)
10         if p Failed then
11             i.UpdateGoals(IMR, γ)
12             MK.UpdateKnowledge(IMR, γ)
13             FU.CheckFailureType(p)
14             return (MK, FU)
15         else if F ← isAchieved then
16             i ← ∅
```

Algorithm 1 shows the Mission Planning process and its interaction with the Run-Time Plan Verbaliser. The system extracts the mission goals (line 2) and initialises the intermediate mission requirements ($\mathcal{IMR}$) with the initial state defined in the mission requirements ($\mathcal{MR}$) (line 3). The online planning algorithm is active until all mission goals are completed (line 4). It takes the current knowledge to generate the planning problem (line 5), which is used to obtain a plan solution (line 6). The algorithm cancels any plan in execution (line 7) before dispatch the new plan generated (line 8). The method keeps checking the updates from SEA (line 9) during the whole mission. If SEA notifies substan-

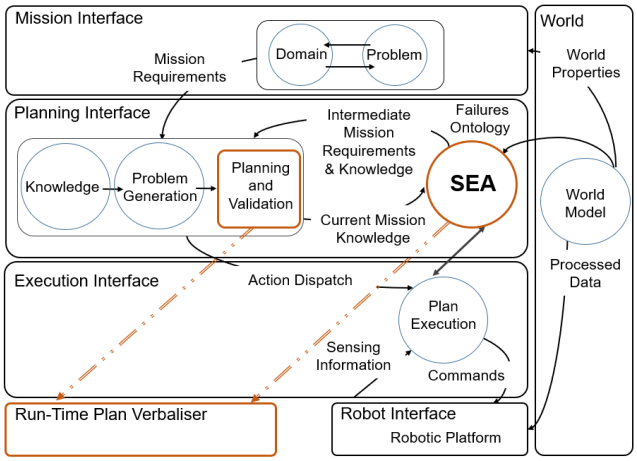


Figure 5: High-level system architecture that combines Online Planning, SEA and Plan Verbaliser components.

tial changes and gives the advice of replanning (line 10), the set of goals i is updated (line 11), removing the global mission goals already implemented. Then the mission knowledge ($\mathcal{MK}$) (line 12) is updated to be used by SEA for plan surveillance. Then the system sends an update regarding the replanning process to the verbaliser (line 13), which is used to describe changes in the original plan to the users. This update contains the type of failure (i.e., localisation failure, battery level, etc.), the failure's risk level (i.e., high, standard and low), and the new goals introduced in the mission to recover the task from the loss before continuing with the execution. The system returns the actual $\mathcal{MK}$ and the information regarding failure update ($\mathcal{FU}$) (line 14), which consider the failure types defined for the underwater domain. On the contrary, if any notification is provided, the system checks when the whole plan is achieved and stops (line 15-16).

Combining the Mission Planning framework and the Run-Time Plan Verbaliser allows the users to clearly understand plan execution in a dynamic environment. This includes the explanation of the reasons for replanning during mission implementation as a consequence of failures. In addition, the verbaliser acts as a bridge to provide information about the addition of new mission goals associated with recovering the robot from failure situations. This is key to maintaining situation awareness of the user and building a mental model of what the system would do in failure scenarios.

6 Combining Planning and Verbalisation

We now define the main elements in the architecture that connect the mission planning and verbalisation process. We introduce a small example that illustrates the steps in the entire pipeline.

System Architecture

Figure 5 shows the system architecture that contains five modules. The elements in this architecture are an extension to the system in Figure 4. However, here we focus on the system dynamic concept, which expands the verbalisation process to more dynamic and realistic solutions. We deploy the proposed framework on a BlueROV2 (see Figure 1). Our BlueROV2 is a full ROS-enabled (Quigley et al. 2009) system. Our planning and verbalisation architecture extends ROSPlan (Cashmore et al. 2015).

The *Mission Interface* includes the PDDL domain and problem. The *Planning Interface* is in charge of generating a solvable plan using all the available knowledge at the planning time. The plan is parsed and dispatch to the Execution Interface. SEA introduces the Intermediate Mission Requirements component $\mathcal{IMR}$ extending the initial Mission Requirements $\mathcal{MR}$ when notifications of failures or substantial changes occur during the plan implementation. SEA receives feedback for the Online Planning and Plan Execution to define the propositions to change and goals to add or remove. Besides, SEA is connected to the World to determine the possible types of failures associated with the environment. SEA can command the Online Planning to cancel the actual plan execution, acknowledging failures for processing the data arriving from the Execution Interface. The *Execution Interface* takes the dispatched action by the Planning Interface and translates its AUV action commands. The interface acts as a bridge between real data acquisition and SEA, helping determine the quality of plan implementation and identifying failure sources. This interface embeds low-level algorithms to support the evaluation of the plan implementation quality. The *Robot Interface* includes the robotic platforms in the mission. Finally, the *Plan Verbaliser* connects with the Online Planning and SEA components to verbalise the implementation of the current plan. This framework provides information to the user about mission failures, including reasons and future behaviours of the robotic platform to maintain the system's survivability.

System Workflow: Example

This example illustrates the data flow when implementing a mission using the Goal-Based Mission Planner and the Planning Verbaliser. Considering the PDDL domain and problem in Section 3, our mission goals are: turn a valve (`valve_turned wp25`) and recover the `auv` at a safe position (`recovered wp20`). The Online Planning system takes the Mission Requirements and implements an initial plan that navigates the `auv` from the initial position at `wp0` to `wp25` to manipulate the valve, then the `auv` should navigate from `wp25` to a safe position marked at point `wp20`. The hardware needs to be stable to implement these goals. This is defined using the predicate (`hardware_stable auv`), defined in the initial state, which has to be true to implement the manipulation or the recovery action. The first action in the plan (`navigate auv wp0 wp25`) is dispatched for execution. The Plan Verbaliser receives the notification of the action in execution, and it verbalises:

- *AUV is moving from point0 to point25.*

The subsequent information the verbaliser provides is associated with successful or unsuccessful action implementation. Supposing the action is executed successfully, the Plan Verbaliser receives updated state information from the Online Planning System, which is verbalised as follows: *AUV*

is now located at point25. Then the next action is dispatched for execution (`manipulation auv wp25 arm`). The verbaliser receives that information and notifies the user:

- *AUV is manipulating a valve at point25.*

Suppose during the implementation of this action, the SEA component receives a notification from a low-level algorithm notifying of a hardware issue affecting the system. SEA evaluates the failure characteristics advising the Online Planning System to cancel the current plan and generate a new plan that makes the system recover from the hardware issue before continuing with the goal implementation. SEA makes updates in the initial knowledge removing the `hardware_stable auv` proposition from the initial state. The Online Planner cancels the plan and takes the current state knowledge to generate a new plan that forces the system to perform a hardware repair. The Plan Verbaliser receives the notification the action `(manipulation auv wp25 arm)` was cancelled, and it enquires the SEA framework for additional information. SEA provides a report regarding the failure type (hardware), risk level (high), and additional specifications (water leak). This intelligence is used to provide further clarification to the user concerning the reasons for plan alterations: first,

- *Manipulation has been cancelled*, and second,
- *A water leak issue detected with the high-risk level.*

After the new plan is generated and the first action is dispatched, the Verbaliser starts the description of the latest action that navigates the `auv` to be repaired. After the `hardware-repair` action, the proposition `(hardware_stable auv)` is reestablished, and the planning system can deal with a solution to solve the uncompleted goals. At the same time, the verbaliser maintains the communication with the user, the Online Planner and SEA.

7 Planning and Verbalisation: Mission

In this section, we present a run-time planning solution for the plan in Figure 2. This run-time planning example evaluates the potentialities of dynamic re-planning when unexpected situations (i.e. substantial changes in the initial model) in the original plan occur that force the system to replan for specific (new) initial conditions. Using this knowledge, the verbaliser can provide truthful information to users by verbalising the plan execution and deviations.

Table 2 shows the plan execution when the original plan is affected by unexpected changes that make the initial solution unsolvable. Therefore, the system experiences plan deviations. The AUV starts the performance of the original plan. Supposing the AUV losses track of the environment features and hence its localisation. The low-level mapping algorithm creates multiple maps of the same structure. SEA receives a notification the AUV is not localised correctly concerning the structure when implementing action `(map auv slam wp13 wp14)`. The Mission Planning strategy receives feedback from the SEA component notifying the failure and proposing a replanning that includes new goals with high priority to localise the AUV in the initial map before implementing the uncompleted original goals. The Run-Time Plan

Run-Time Plan (1)	**Description**
`(navigation auv wp0 wp10)`	Navigate to `wp10`
`(map auv slam wp10 wp11)`	Create map `wp10` to `wp11`
`(map auv slam wp11 wp12)`	Create map `wp11` to `wp12`
`(map auv slam wp12 wp13)`	Create map `wp12` to `wp13`
`(map auv slam wp13 wp14)`	Create map `wp13` to `wp14`
SEA Notification	AUV Localisation Issue
Run-Time Plan (2)	**Description**
`(map auv slam wpra wpr1)`	Localise `auv`
`(map auv slam wpr1 wpr2)`	Localise `auv`
`(map auv slam wpr2 wpr3)`	Localise `auv`
SEA Notification	AUV Localised
Run-Time Plan (3)	**Description**
`(map auv slam wpra wp14)`	Create map `wpra` to `wp14`
`(map auv slam wp14 wp15)`	Create map `wp14` to `wp15`
`(map auv slam wp15 wp16)`	Create map `wp15` to `wp16`
`(map auv slam wp16 wp10)`	Create map `wp16` to `wp10`
`(navigation auv wp10 wp0)`	Navigate to `wp0`
`(navigation auv wp0 wp27)`	Navigate to `wp27`
SEA Notification	AUV Hardware Issue
Run-Time Plan (4)	**Description**
`(navigation auv wpra wps)`	Navigate to surface
`(communicate auv wps)`	Communicate with base
`(hardware-repair auv wps)`	Repair hardware
`(navigation auv wps wp27)`	Navigate to `wp27`
`(navigation auv wp27 wp26)`	Navigate to `wp26`
`(inspect-rock auv wp26)`	Inspect rock at `wp26`
`(navigation auv wp26 wp25)`	Navigate to `wp25`
`(manipulate auv wp25)`	Manipulate valve at `wp25`
`(navigation auv wp25 wp20)`	Navigate to `wp20`
`(recover auv wp20)`	Recover `auv` at `wp20`

Table 2: A run-time plan implementation. The original temporal plan and the recovery plans after failures.

(2) makes the robot navigate to a set of points previously defined by a low-level algorithm (focus in robust relocalisation) where there is a high probability the AUV manages to relocalise. SEA identifies the type of failure and reasons for the alternative plan that enables the robot to recover. During the execution of the recovery plan, the AUV can relocalise at anytime. In that case, SEA determines the recovery plan is completed. The system will then call the original plan's uncompleted goals and send these new requirements to the Online Planner to obtain a plan that solves the remaining goals. Suppose the AUV completes the recovery plan (intermediate), and relocalisation is not achieved. In that case, the system will ask for new relocalisation points. Run-Time Plan (3) shows that SEA notifies that the AUV is localised.

Our approach can deal with other types of failures, such as battery level notifications and hardware issues (e.g., thruster failures, water leaking, etc.). One example of these situations is presented in Run-Time Plan (3). SEA notifies that there is a hardware problem (water leak) that is affecting the AUV. The system forces the generation of a plan repair that makes

Run-Time P. (1)	
Action Status	**Explanation**
T0: AE	AUV is moving from point0 to point10.
T1: AA	AUV is now located at point10.
...	
T0: AE	AUV maps area between point10 and point11.
T1: AA	Mapping has been completed.
T0: AE	AUV maps area between point13 and point14.
T1: CA	Mapping was cancelled.
AUV Loc. Issue Detected	A localisation error has been detected with standard risk. Alternative plan is being devised.
Run-Time P. (2)	
Action Status	**Explanation**
T0: AE	AUV is re-localising.
T1: AA	Localisation has been completed.
...	
AUV Localised	Localisation completed AUV has been localised.
Run-Time P. (3)	
Action Status	**Explanation**
T0: AE	AUV maps area between pointA and point14.
T1: AA	Mapping has been completed.
...	
T0: AE	AUV is moving from point0 to point27.
T1: CA	Navigation to point27 was cancelled.
AUV Hardware Issue Detected	A water leak issue has been detected with high risk level. Alternative plan is being devised.
Run-Time P. (4)	
Action Status	**Explanation**
T0: AE	AUV is moving from current point to surface.
T1: AA	AUV is located at surface.
T0: AE	AUV is communicating at surface.
T1: AA	AUV has communicated with C2.
T0: AE	AUV is getting repaired at surface.
T1: AA	AUV has been repaired.
T0: AE	AUV is moving from surface to point27.
T1: AA	AUV is now located at point27.
...	
T0: AE	AUV is manipulating a valve at point25.
T1: AA	AUV has turned the valve.
T0: AE	AUV is moving from point25 to point20.
T1: AA	AUV is now located at point20.
T0: AE	AUV is being recovered at point20.
T1: AA	AUV is recovered.

Table 3: Verbalisation for a run-time planning solution. Status provides information to recognise if the action is enabled (AE) achieved (AA) or cancelled (CA). T0 and T1 represent the action start and end times.

the AUV navigate to the surface, communicate the problem, and implement the repair before executing the uncompleted actions in the plan. Run-Time Plan (4) shows the sequence of actions the AUV implements to deal with this situation. Notice, the system advises the online planning module to implement a plan repair that forces the robot to repair the hardware. SEA proposes an update of the current knowledge that removes the proposition associated with the hardware state `(hardware_stable ?r - robot)` considering the hardware issues. The most recent state is used to generate a new plan that deals with this situation by navigating the AUV to a surface point where maintenance is accomplished before executing the rest of the goals. The action `hardware-repair` reestablishes the predicate associated with hardware stability. SEA helps to avoid situations where the model inaccuracy leads to build preliminary plans.

Table 3 describes the verbalisation of the plan execution in Table 2. We select the interesting actions in the plan (actions in red) for verbalisation. In addition, we introduce the verbalisation of all failure notifications. The action status provides the information regarding the action state, which can be enabled (AE), achieved (AA) or cancelled (CA). This status is evaluated at the action start (T0) and the end (T1). We use this information to provide an idea of the mission time to the user. The verbalisation for each action is presented on the right side of Table 3. When an error is detected, the specific type, risk level, and additional specifications are mentioned. The explanations make the user aware that the robot will change its behaviour with respect to the original plan. Once replanning is completed, the Verbaliser notifies the user. The system assures the user the mission is completed when goals are achieved and the robot is retrieved. Inner functionalities such as the low-level algorithms to identify failures are not described during plan execution to avoid mental overload and loss of interest.

8 Conclusion and Future Work

We have introduced a planning verbaliser with our work, which offers state awareness to users/operators with minimal experience on AI planning algorithms through natural language explanations. Our system reasons about problem model and plans by analysing the offline solution. During plan execution, it gives details about action completion, error detection and mission replanning. The system shows effectiveness in defining the reasons for plan deviations during mission execution. Further evaluation is required to estimate the efficacy of the explanations towards the disambiguation of decision making. Future work aims to trigger API calls via dialogue-based interaction to enable the user to request specific information about missions. Additionally, we would like to personalise explanations to accommodate both expert and novice planning users and maintain interest in the interaction by presenting the optimal amount of information. Finally, we intend to automate content selection with data from a generic domain and ontology, which will include multiple vehicle types and actions.

Acknowledgments

This work was funded and supported by the EPSRC ORCA Hub (EP/R026173/1), UKRI Node on Trust (EP/V026682/1), EPSRC CDT on Robotics and Autonomous Systems (EP/S023208/1), SeeByte Ltd and SRPe.

References

Adadi, A.; and Berrada, M. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE Access* 6: 52138–52160.

Babli, M.; Onaindia, E.; and Marzal, E. 2019. Extending planning knowledge using ontologies for goal opportunities. *arXiv preprint arXiv:1904.03606* .

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of ICAPS*, 2–10.

Bouillet, E.; Feblowitz, M.; Liu, Z.; Ranganathan, A.; and Riabov, A. 2007. A Knowledge Engineering and Planning Framework based on OWL Ontologies. In *Proc. of ICKEPS*.

Carreno, Y.; Pairet, È.; Petillot, Y.; and Petrick, R. P. A. 2020. A decentralised strategy for heterogeneous auv missions via goal distribution and temporal planning. In *Proceedings of ICAPS*, 431–439.

Carreno, Y.; Scharff Willners, J.; Petillot, Y. R.; and Petrick, R. 2021. Situation-Aware Task Planning for Robust AUV Exploration in Extreme Environments. In *Proceedings of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild*.

Cashmore, M.; Collins, A.; Krarup, B.; Krivic, S.; Magazzeni, D.; and Smith, D. 2019. Towards explainable AI planning as a service. In *Proceedings of the ICAPS Workshop on Explainable Planning (XAIP)*, 104–112.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *Proceedings of ICAPS*, 333–341.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The emerging landscape of explainable automated planning & decision making. In *Proceedings of IJCAI*, 4803–4811.

Cioffi, M.; and Thompson, S. 2007. Planning with the Semantic Web by fusing Ontologies and Planning Domain Definitions. In *Proc. of Conference on Innovative Techniques and Applications on Artificial Intelligence*, 289–302.

Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20: 61–124.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. In *Proc. of IJCAI Workshop on Explainable AI*.

Galanis, D.; and Androutsopoulos, I. 2007. Generating multilingual descriptions from linguistically annotated OWL ontologies: the NaturalOWL system. In *Proc. ENLG*, 143–146.

Garcia, F. J. C.; Robb, D. A.; Liu, X.; Laskov, A.; Patron, P.; and Hastie, H. 2018. Explainable autonomy: A study of explanation styles for building clear mental models. In *Proceedings of the INLG*, 99–108.

Gatt, A.; and Reiter, E. 2009. SimpleNLG: A realisation engine for practical applications. In *Proc. of ENLG*, 90–93.

Glimm, B.; Horrocks, I.; Motik, B.; Stoilos, G.; and Wang, Z. 2014. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning* 53(3): 245–269.

Hastie, H.; Liu, X.; and Patron, P. 2016. A demonstration of multimodal debrief generation for AUVs, post-mission and in-mission. In *Proceedings of ICMI*, 404–405.

Hastie, H.; Liu, X.; and Patron, P. 2017. Trust triggers for multimodal command and control interfaces. In *Proceedings of ICMI*, 261–268.

Lamy, J.-B. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine* 80: 11–28.

McCluskey, T. L.; and Cresswell, S. N. 2005. Importing Ontological Information into Planning Domain Models. In *Proceedings of the ICAPS Workshop on the Role of Ontologies in Planning and Scheduling*.

McGuinness, D. L.; Van Harmelen, F.; et al. 2004. OWL web ontology language overview. *W3C recommendation* 10(10): 2004.

McNeill, F.; and Bundy, A. 2007. Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *IJSWIS* 3(3): 1–35.

Miguelanez, E.; Patron, P.; Brown, K. E.; Petillot, Y. R.; and Lane, D. M. 2010. Semantic knowledge-based framework to improve the situation awareness of autonomous underwater vehicles. *IEEE Transactions on Knowledge and Data Engineering* 23(5): 759–773.

Moon, J.; Magazzeni, D.; and Cashmore, M. 2019. Towards Explanations of Plan Execution for Human-Robot Teaming. In *Proceedings of SDMM*, 58–64.

Nikolaidis, S.; Kwon, M.; Forlizzi, J.; and Srinivasa, S. 2018. Planning with verbal communication for human-robot collaboration. *ACM Transactions on HRI* 7(3): 1–21.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source Robot Operating System. In *Proceedings of the ICRA Workshop on Open Source Software*.

Robb, D. A.; Chiyah Garcia, F. J.; Laskov, A.; Liu, X.; Patron, P.; and Hastie, H. 2018. Keep me in the loop: Increasing operator situation awareness through a conversational multimodal interface. In *Proceedings of ICMI*, 384–392.

Rosenthal, S.; Selvaraj, S. P.; and Veloso, M. M. 2016. Verbalization: Narration of Autonomous Robot Experience. In *Proceedings of IJCAI*, 862–868.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *Proceedings of AAAI*.

Sridharan, M.; and Meadows, B. 2019. Towards a Theory of Explanations for Human-Robot Collaboration. *KI-Künstliche Intelligenz* 33(4): 331–342.

Tenorth, M.; and Beetz, M. 2009. KnowRob—knowledge processing for autonomous personal robots. In *Proceedings of IROS*, 4261–4266.

Thielstrom, R.; Roque, A.; Chita-Tegmark, M.; and Scheutz, M. 2020. Generating explanations of action failures in a cognitive robotic architecture. In *Workshop on Interactive Natural Language Technology for Explainable AI*, 67–72.