Noname manuscript No.
(will be inserted by the editor)

Interacting with Team Oriented Plans in Multi-Robot Systems

**Alessandro Farinelli, Masoume M. Raeissi,
Nicolo' Marchi, Nathan Brooks, Paul Scerri**

the date of receipt and acceptance should be inserted later

Abstract Team oriented plans have become a popular tool for operators to control teams of autonomous robots to pursue complex objectives in complex environments. Such plans allow an operator to specify high level directives and allow the team to autonomously determine how to implement such directives. However, the operators will often want to interrupt the activities of individual team members to deal with particular situations, such as a danger to a robot that the robot team cannot perceive. Previously, after such interrupts, the operator would usually need to restart the team plan to ensure its success. In this paper, we present an approach to encoding how interrupts can be smoothly handled within a team plan. Building on a team plan formalism that uses Colored Petri Nets, we describe a mechanism that allows a range of interrupts to be handled smoothly, allowing the team to efficiently continue with its task after the operator intervention. We validate the approach with an application of robotic watercraft and show improved overall efficiency. In particular, we consider a situation where several platforms should travel through a set of pre-specified locations, and we identify three specific cases that require the operator to interrupt the plan execution: i) a boat must be pulled out; ii) all boats should stop the plan and move to a pre-specified assembly position; iii) a set of boats must synchronize to traverse a dangerous area one after the other. Our experiments show that the use of our interrupt mechanism decreases the time to complete the plan (up to 48% reduction) and decreases the operator load (up to 80% reduction in number of user actions). Moreover, we performed experiments with real robotic platforms to validate the applicability of our mechanism in the actual deployment of robotic watercraft.

Alessandro Farinelli, Masoume M. Raeissi, Nicolo' Marchi
Computer Science Department, University of Verona
Verona, Italy, CAP 37134
E-mail: alessandro.farinelli@univr.it, masoume.raeissi@univr.it, marchi.nicolo@gmail.com

Nathan Brooks
Robotics Institute, Carnegie Mellon University
Pittsburgh, PA USA
E-mail: nbb@cs.cmu.edu

Paul Scerri
Platypus, LLC
Pittsburgh, PA USA
E-mail: paul@senseplatypus.com

1 Introduction

Robotics technology has matured sufficiently to make the idea of building robot teams for real environments a serious possibility. For applications ranging from disaster response [2, 30, 16] to environmental monitoring [8, 27] to surveillance [9], approaches are emerging to allow small numbers of humans to control teams of robots to achieve complex objectives. An effective way of doing this is via team plans [24, 11, 32], which allow an operator to interact via high level objectives and use automation to work out the details. For example, a plan for environmental monitoring might tell robots to collect a certain type of information in a certain area, leaving the robots to work out who goes where to collect the information.

However, in most real domains, human operators will occasionally need to directly control a robot for some purpose, perhaps to protect a robot from a danger it cannot perceive or to achieve some specific objectives that the robot is not capable of understanding. Typically, when a robot plan is interrupted, any team plan that the robot was participating in will be terminally impacted. In some cases, the rest of the team can reorganize without the interrupted robot and then reorganize when the interruption is over, but this depends on the plan, the particular situation, and nature of the interruption. In general, how to respond to an external interruption is very sensitive to the specific context of the plan and if the context is not taken into account when dealing with the interruption, overall performance will be poor.

In this paper, we are specifically looking at a domain of teams of robotic boats collecting information on bodies of water [19]. In such applications, one or a small number of experienced operators, perhaps water scientists, are managing between five and twenty five boats on a body of water. Large manned boats and water phenomena are external dangers to the robots that the human operators might be able to help mitigate. In other cases, operators might have some external knowledge about what is going on in the water that allows them to direct resources in a very specific way to get very specific information. Hence, while it is necessary to utilize team plans to make use of multiple assets, interruptions are an inevitable and important part of operations.

For example, consider a situation where the team of boats is instructed to acquire measurements in a set of pre-specified locations. Each boat is assigned to a subset of such locations and all boats execute their plan in parallel. If one of the boats must be pulled out from the plan (e.g., to recharge the battery), the other boats should continue their plan without stopping. In another situation, the operators might want to slightly change the course of actions of the entire team (e.g., reassign tasks to all available boats when one is pulled out) or even drastically change the current plan of all boats to handle a dangerous situation (e.g., a manned boat suddenly enters the area of operation). The key focus of this paper is to provide a general mechanism to handle all the above situations without aborting and restarting the current plan

To realize these sophisticated interactions, we adopt an approach for creating team plans with Petri Nets that allow specification of complex, parallel, and hierarchical plans. To deal with external interruptions, we add functionality used to indicate the start of two categories of interruptions without providing knowledge of the nature of the interruption. The approach allows transitions to be created from any place in the Petri Net to a place that waits for the interruption handling to be completed before sending the plan back to an appropriate place. Depending on the nature and timing of the interaction, relative to the specific context of the plan, the expressive approach allows for a range of possibilities to be encoded, including restarting the plan, directly resuming, or going through some intermediate steps to restart effectively. The key is that the plan designer can work out in advance how to handle interruptions at a particular place in the plan and encode efficient and effective resumptions.

In some interesting plans, there are specific features for specific assets. Such features could be capabilities of the robots (e.g., a boat equipped with particular sensing devices) or roles that the robots play for the team plan (e.g., a set of boats exploring a specific area). Our team plan approach captures this by using a Colored Petri Net formalism that differentiates the various assets by using colored tokens. In this way, we can precisely control which assets should be interrupted when specific incidents occur and the plan can be designed to react differently to different assets being interrupted. For example, an operator might want to interrupt plan execution only for a specific subset of the robots, e.g., the ones that are entering a dangerous area, while leaving the plan for the rest of the platforms unchanged. The key is that the design approach provides the representational power to handle such interruption effectively.

To evaluate the approach, we use a simulator of the robot boats and a novel technique for determining the value of the concept. The simulator is used for all development and testing of algorithms on the real robots, hence it is an accurate simulator of their capabilities. However, it is very difficult to design an experiment with real human operators where interrupts would be distributed over the whole length of the plan and vary in length in non-trivial ways. Moreover, this would require a proper evaluation of the operator interface which falls outside the scope of this contribution. Instead, our experimental approach simulates all possible interrupts multiple times. While in practice interrupts at some times might be more common than others, the concept of the approach is that any interrupt is handled smoothly hence the experimental setup provides a significant indication of the power offered by the interrupt mechanism without considering the skills of the human operators. In more detail, we consider a cooperative location visit plan, where a set of interesting locations, selected by the user, must be visited by a set of platforms to perform point measuring tasks. We identify three typical incidents that require the operator to interrupt the normal execution of the team oriented plan: i) a boat must be pulled out (e.g., because it is running out of battery); ii) all boats should stop the plan an reach a pre-specified assemble position (e.g., to avoid colliding with a manned boat that entered the area of operation); iii) a set of boats must synchronize to traverse a dangerous area one after the other, so that the human operator can closely monitor the behavior of each single boat and tele-operate the platform if necessary. For each of these incidents we compared the execution of team plans without specific interrupt handling to plans where interrupt handlers were explicitly encoded by using our framework and we found significant improvement in overall efficiency, i.e., up to a 48% reduction in time to complete a plan and up to a 80% reduction in operator load. Moreover, to validate the use of our interrupt mechanism when deploying robotic watercraft, we performed various experiments with real platforms. Such experiments indicate that our mechanism can be effectively used in actual operations.

The rest of the paper is organized as follows, Section 2 provides necessary background on Colored Petri Net (CPN) and plan monitoring for multi-agent systems. Section 3 describes the robotic boat system we considered here and the plan specification language used in such system. Section 4 describes the interrupt mechanism we propose and Section 5 details our empirical methodology discussing obtained results. Finally, Section 6 concludes the paper.

2 Background and Related Work

In this section we will first provide necessary mathematical background on Petri Net and Colored Petri Net and then discuss related work on plan monitoring in multi-agent systems.

2.1 Petri Net

Petri Net (PN) is a widely used mathematical and graphical modeling tool for describing concurrency and synchronization in distributed systems. Graphically, a PN is represented by a directed bipartite graph, in which nodes could be either places or transitions, arcs connect places to transitions and vice versa. Places in a Petri net contain a discrete number of marks called *tokens*. A particular allocation of tokens to places is called a *marking* and it defines a specific state of the system that the Petri Net represents.

Formally, a PN is a tuple $PN = (P, T, F, W, M_0)$, where $P = \{p_1, p_2, \ldots, p_m\}$, is a finite set of places; $T = \{t_1, t_2, \ldots, t_n\}$, is a finite set of transitions; $F \subseteq (P \times T) \cup (T \times P)$, is a set of arcs; $W : F \to \mathbb{N}$, is a weight function; $M_0 : P \to \mathbb{N}_0$ is an initial marking.

The marking of a PN evolves based on the firing behavior of the transitions. A transition t can fire whenever it is enabled (e.g., when each input place p_i of the transition is marked with at least $W(p_i, t)$ tokens) and if the transition fires $W(p_i, t)$ tokens are removed from each input places p_i and $W(t, p_j)$ tokens are added to each output place p_j.

Colored Petri Nets (CPN) [10] extend Petri Nets where tokens become more informative and each of them has attached a data value called the token *color*. The firing behaviors of transitions and consequently the evolution of the markings now depend on the colors of the tokens. In particular, tokens can now be identified and related to specific agents, thus providing a compact and convenient modeling language for team oriented plans.

Moreover, similar to PN, CPN can be analyzed and verified either by means of simulation or formal analysis methods[1], thus allowing validation of team oriented plans before their execution.

2.2 Plan monitoring in multi-agent systems

The problem of monitoring plan execution in agents and multi-agent systems is a key issue when such systems must be deployed in real-world applications where the environment is typically dynamic and action execution is non deterministic. Consequently, over the years plan monitoring has become a crucial research topic that has been addressed from several different perspectives [28, 21, 29]. However, here we focus on approaches that are most relevant to our work as they are explicitly designed for multi-agent or multi-robot systems. Hence, we discuss architectures for Human-Robot coordination [30, 16] and plan specification [24, 11], approaches for *Adjustable Autonomy* in Multi-Agent Systems, and plan monitoring approaches based on Petri Nets [12, 32].

Human-Robot coordination and mission control are crucial topics for the effective deployment of Multi-Robot Systems in practical applications such as Urban Search And Rescue (USAR) [16, 30] or surveillance [9]. Consequently there has been a significant amount of work targeted at developing software tools and architectures that allow few human operators to command and control a team of (possibly heterogeneous) robots. For example, [16] focuses on methodologies to facilitate teamwork between humans and robots involved in USAR applications. Specifically, they propose the application of a general Multi-Agent System architecture (i.e., RETSINA [23]) to a Multi-Robot System. Similar to our work they focus on a methodology to allow a few human operators to monitor the activities of a team of robots that operate with a high degree of autonomy. In contrast, our focus is on a specific approach for plan specification (i.e., CPN) and on an interrupt mechanism that

[1] See for example CPN *Tools* [18]

allows human operators to smoothly change the behavior of a team plan without terminally impacting the whole plan execution.

Two successful BDI-based frameworks for plan specification are STEAM and BITE, which enable a coherent teamwork structure for multiple agents. The key aspect of STEAM [24] is team operators, which are based on the Joint Intentions Theory introduced by [3]. In STEAM, agents can monitor the team's performance and reorganize the team based on the current situation. BITE, which was introduced by Kaminka & Frenkel [11], specifies a library of social behaviors and offers different synchronization protocols that can be used interchangeably and mixed as needed. However, while both these works provide key contributions for building team oriented plans, they do not provide any specific mechanism for a human operator to interrupt the execution of such plans.

In this perspective, an important strand of research focuses on the concept of *Adjustable Autonomy*, where agents can vary their level of autonomy and transfer decision-making control to humans (or other agents) [4,22,20]. The main focus for adjustable autonomy approaches is to facilitate teamwork when autonomous agents must cooperate with themselves and with humans: a key issue in this setting is to devise effective techniques to decide whether a transfer of control should occur and when this should happen. For example, Scerri and colleagues in [20] propose the use of transfer of control *strategies* which are conditional sequences of two types of actions: transfer of decision making control (e.g., an agent giving control to a user) and coordination changes (i.e., an agent delaying the execution of a joint task). The authors propose an approach based on Markov Decision Processes to select an optimal strategy and evaluate their method in a deployed Multi-Agent System where autonomous agents assist a group of people in daily activities (e.g., scheduling and re-scheduling meetings, ordering meals, and so forth). Similar to our work, adjustable autonomy techniques (and transfer of control strategies in particular) focus on the interaction between human users and Multi-Agent Systems, where agents have a high degree of autonomy. However, rather than providing a general framework to devise transfer of control strategies, our approach focuses on a specific, important case of transfer of control, where a human decides to intervene and change the execution of a team oriented plan. Our contribution is then to propose a plan representation framework (based on Colored Petri Nets) and a specific interrupt mechanism to maximize efficiency and reduce operator load in this specific setting.

There is substantial literature on the topic of using Petri Nets [17] and variants such as Colored Petri Nets [10] as the basis for representing team plans. Similar to state machines and other directed graphs, Petri Nets give an intuitive view of the plan, but provide additional power useful for multi-robot teams, such as synchronization and concurrency. Moreover, there are several analysis methods for Petri Nets [15] [1] which can test different properties, such as reachability, boundedness, liveness, reversibility, coverability, and persistence. These methods allows for finding errors before the testing phase on simulated or physical platforms hence providing a significant help to the system designers. Consequently, there are several approaches proposing the use of Petri Nets for representing team plans, such as Petri Net Plans [32], task plan representation by Petri Nets [5], Task Petri Nets [26], Agent Petri Nets [14], and PrT nets [31]. In addition, Colored Petri Nets have also been considered for representing multi-robot plans [13]. These approaches can represent complex team plans that include coordination and synchronization among the robots. However they do not explicitly consider the involvement of human operators and their intervention in case of possible robot failures or unexpected events.

In more detail, one of the first approaches based on Petri Net for multi-agent plan monitoring was proposed by King et al. [12]. In particular they propose the use of automated

planning methods to generate plans for multiple agents. Such plans are then compiled into Petri Nets for analysis, execution and monitoring. The approach handles possible failures during plan execution by re-planning at run-time. In particular, the agent which experienced a problem/failure informs the human operator, who starts the re-planning process. Hence, the human operator intervention happens only after receiving a request from the agent; moreover, re-planning at run-time might be problematic for applications that must operate within real-time constraints.

Recently, Ziparo et al. proposed an approach for plan monitoring called Petri Net Plan (PNP) [32]. PNP takes inspiration from action languages and offers a rich collection of mechanisms for dealing with action failures, concurrent actions and cooperation in a multi-robot context. One important functionality offered by the formalism of PNP is the possibility to modify the execution of a plan at run-time using interrupts. Our work also focuses on interrupting multi-robot plans; however, here we are mainly interested in human operators interrupting the normal execution of a team-plan rather than action failures and plan synchronization. In more detail, here we take a different approach for plan representation as we encode team plans by using Colored Petri Nets in contrast to what is proposed in [32] where a team-plan is a collection of several single-agent plans represented with standard Petri Nets. This is a significant difference as it allows us to represent plans involving several agents with a very compact structure as agents are represented by the colored tokens and not explicitly in the network. Moreover, by using CPN we can represent different types of interrupts, i.e., team-level and platform specific (see Section 4) thus providing a rich model to allow sophisticated interactions between the human operators and team plans.

Note that an initial version of this paper appeared in [7]. However, here we provide more results on the performance of our approach by considering a new situation where boats must synchronize their behaviors to traverse a dangerous area one at a time. Moreover, we validate our interrupt mechanism by performing experiments with real robotic platforms.

3 The cooperative robotic watercraft framework

This work focuses on a system of robotic boats developed as part of the Cooperative Robotic Watercraft project [19]. In this section we describe the team plan specification language used in the system and provide a brief overview of the whole system.

3.1 SAMI Petri Net

SAMI Petri Nets (SPN) are based on Colored Petri Nets and Hierarchical Petri Nets, with various extensions to add the capability to send and receive commands and information from team members, to perform and reference task allocations, and to capture situational awareness and mixed initiative (SAMI) directives. In more detail, SPNs are based on the CPN modelling language defined in [10], which supports hierarchical CPN and the use of variables.

We define an SPN structure as the following tuple $\langle P, T, F, E, R, SM \rangle$, where:

- $P = \{p_1, p_2, ..., p_i\}$ is a finite set of *places*.
- $T = \{t_1, t_2, ..., t_j\}$ is a finite set of *transitions*.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *edges*.
- $E = \{e_1, e_2, ..., e_k\}$ is a set of *events*.

- $R = F \rightarrow \{r_1, r_2, ..., r_m\}$ is a mapping of *edges* to a set of *edge requirements*.
- $SM = P \rightarrow \{sm_1, sm_2, ..., sm_l\}$ is a mapping of *places* to a set of *sub-missions*.

The SPN models the execution of a team plan by representing the current state of the system (i.e., the markings of the places), the evolution of system states over time, and the interactions between the different components of the systems. In more detail, the SPN implementation defines a *plan manager*, which is an execution engine responsible for all interactions among the different components of the robotic platforms. All interactions take the form of commands (or requests) sent from the plan manager to the robotic platforms (or to the operators) and information received from human operators/robotic platforms.

While we will use the SPN framework in our application domain (i.e., cooperative robotic watercraft), we make no context specific assumptions for the team plan specification language (and for the interrupt mechanism we will define in Section 4). Hence our approach can be use in other scenarios where human operators should design and monitor team plans for multi-robot systems.

In what follows, we describe each of the main elements of the SPN and then provide operational semantics in the form of firing rules for the transitions.

Events: events fall under two categories: *output events* and *input events*.
Output events are associated to places in the Petri Net (using the mapping $E_O = P \rightarrow \{oe_1, oe_2, ..., oe_q\} \subseteq E$, which maps each place to a set of output events) and represent commands or requests that are sent to human operators, robot proxies[2], or agents. When a token(s) enters a place, all the output events on the place, $E_O(p)$, are processed. The registered handler for that class of output event is sent the output event oe along with the tokens that just entered the place (Algorithm 3).

For output event classes that contain data fields, there are 3 ways to specify the information, which are listed here with example usage in our outlined scenario: (1) Value defined offline by the Petri Net developer (the battery voltage threshold to send a low-energy alert to the operator). (2) Value defined by the operator at run-time (a safe temporary position for robots to move to in order to avoid an incoming manned boat). (3) Variable name whose value is written by an input event at run-time (a variable to retrieve the path returned from a path planning agent via a "Path Planning Response" input event). Variables are explained in more depth later in this section.
Input events are associated to transitions (using the mapping $E_I = T \rightarrow \{ie_1, ie_2, ..., ie_h\} \subseteq E$, which maps each transition to a set of input events) and contain information received from human operators, robot proxies, or agent services, which perform assistive functions such as path planning, task allocation, and image processing. The set of input events on a transition, $E_I(t)$, are responses to an output event on a place preceding the transition. For an input event ie that will contain information at run-time (such as a generated path or selection from an operator), a variable name is used so the information can be accessed by output events.

Input events contain "relevant proxy" and "relevant task" fields, which contain the identities of the proxy(s) or task(s) (if any) that sent or triggered the input event.

Events in SPN have a function that is very similar to actions in the PNP framework [32]: the PNP framework describes the evolution of a robotic system where states change due to actions and SPN describes the evolution of a team plan where the states change due to events. However, an important structural difference is that in PNP actions are associated to transitions, while in SPN we associate output events to places and input events

[2] With the term *proxy* we refer to a software-service that connects a specific boat with the rest of the system

to transitions. The rationale behind this choice is twofold: first, we have a more compact SPN, second, this results in a more efficient implementation. To see this, consider the place with output event "ProxyExecutePath" in Figure 1 which is connected to a transition with "ProxyPathCompleted". This path execution sequence is captured with one place and one transition. If we instead associate output events with transitions, we would need a place representing the precondition for starting ProxyExecutePath, a transition that actually sends the ProxyExecutePath, a second place that represents that the proxies are executing the path, and a second transition with the ProxyPathCompleted input event. This extra place and transition for each action sequence results in a much less compact network. In addition, we use the output event instance's unique id as criteria for matching a received input event to a transition in the SPN. This is necessary in the common case where an input event is used in multiple transitions, such as having instances of ProxyExecutePath and ProxyPathCompleted, so that the correct transition's firing requirements are updated. In contrast, associating output events to transitions would make the pre-conditions and post-conditions for the events more visible in the CPN representation. This could be a valuable feature for a designer and would be more in line with traditional PN specifications of control systems. However, a precise assessment of this trade-off requires further investigations while our focus here is to provide a mechanism for smoothly handling interrupts in the SPN plan specification language. Hence, we leave the analysis of this issue to future work.

When an input event is received by the system and matched to its corresponding transition in a Petri Net, it is marked as being "received" (Algorithm 1). When a transition fires, its input events' "received" statuses are reset (Algorithm 2).

Variables: Similar to the model for CPN proposed in [10], SPNs support a variable database, where variables are typed and scoped globally or locally. The use of variables is a key element to keep the network compact and to make the plan specification framework flexible and easy to use. Global scope variables allow plans to share information, such as a sensor mapping density, while local scope variables allow multiple copies of a plan to run simultaneously without overwriting instance specific data, such as locations to visit. ifferent variables can be defined for each input event. Fields in output events can refer to these variables, provided they are of the corresponding type and within scope.

Tokens: In general, the CPN modeling language allows to define a variety of color sets for tokens in order to support different data types such as list, structure, enumeration, etc. We now explains our data types for SPN. The SPN tokens have four pieces of information: a name (String), a token type (TokenType), a proxy (ProxyInt), and a task (Task). Each token tk is one of three TokenTypes: *Generic* tokens have no defined proxy nor task and are used as counters. *Proxy* tokens contain a proxy but no task. These are created whenever a robot proxy is added to the system at run-time. *Task* tokens contain a task and might contain a proxy. Task tokens are created by the Petri Net execution engine when a plan is started, creating one for each task in the plan. When the task is allocated to a proxy, the proxy field of the task's corresponding token is set to the proxy assigned to the task. The data within the token (ProxyInt for proxy tokens, Task for task tokens) can be used by events to address specific resources in the team (e.g., tell Proxy A to go to a location or tell Task A it is complete). The data can also be used in edge requirements to require specific proxy or task tokens to be in a place in order for a transition to fire (equivalent to arc inscriptions in CPN Tool). In this sense, the proxy token for Proxy A and the proxy token for Proxy B are of different color sets. The full color set would thus be generic, the list of all proxies, and the list of all tasks. Representing proxies and tasks in this manner allows for multi-robot plans with arbitrary

numbers of team members that must execute the same actions (i.e., the *Proxy Execute Path* in the CLV plan reported in Figure 1) to be constructed and visualized compactly, compared to having an individual Petri Net for each member of the team.

Edge Requirements: *Edges* fall under two categories: *incoming edges* $if \in F$, which connect a place to a transition, and *outgoing edges* $of \in F$, which connect a transition to a place. Similarly, *Edge Requirements* have two categories: *incoming requirements* ir, which are mapped by R from *incoming edges*, and *outgoing requirements* or, which are mapped by R from *outgoing edges*. In a standard Petri Net, incoming edges have a weight which specifies the number of tokens required for a transition to fire, which are then consumed, and outgoing edges have weights which specify the number of tokens to add to the connected place.

Colored Petri Nets allow edges to specify different quantities for different colors of tokens. SPN edge requirements have additional options to maintain the network as compact as possible.

Each **incoming requirement** ir on an *incoming edge* if, $R(if)$, specifies tokens that must be present or absent in the connected place in order for the connected transition to fire. However, when a transition fires, these tokens are not always removed as this could cause undesired interruption of behavior controlled by output events in the connected place. Instead, each **outgoing requirement** or on an *outgoing edge* of, $R(of)$, specifies tokens that should be removed from the incoming places (the places preceding the connected transition) and tokens that should be added to the connected place.

This is achieved by having each outgoing requirement specify a set of tokens and an action to perform on those tokens: take, consume, or add. Taking a token removes it from incoming places and adds it to the outgoing place. Consuming a token removes it from incoming places. Adding a token adds a copy of the token to the connected place. The take action represents the standard operation that is executed on PN and CPN when a transition fires. However, *consume* and *add* are extensions to the standard semantics of PN used in SPN only to maintain the network's compactness. Specifically, the motivation for using these actions is that since we have output events associated to places, we need a way to move a token from a preceding place to a following place without removing it from the initial place. If we expand the network as described above (i.e., adding two places and one transition) we would not need this extension. Furthermore, while we could use standard PN structures to implement these actions (e.g., we could add a specific transition without outgoing edges to consume a token from a place) this would defeat the purpose of having a compact network. Similar to Colored Petri Nets, the set of tokens specified by an edge requirement can be generic tokens or specific task tokens. Edge requirements can also refer to "relevant tokens" which are defined by the input events on the transition being evaluated. The list could contain proxy token(s), in the case of a "Path Completed" input event which specifies the proxy token for the robot that finished, so that at run-time that proxy token can be moved forward in the Petri Net. It could also contain task token(s), in the case of a "Task Completed" input event signaling that a particular task has been completed.

Sub-missions: The SPN language supports hierarchical team plans, allowing a place (called a *sub-mission place*) to have a set of "sub-missions", $SM(p)$. Sub-missions in SPN provide a specific implementation of hierarchical CPN [10]. Each sub-mission sm is an SPN which is run in either *dynamic* or *static* mode. For **dynamic** sub-missions, when tokens enter the *sub-mission place* of the parent plan a new instance of the sub-mission SPN is started and the initial marking is defined as those tokens in the sub-mission's start place (Algorithm 3).

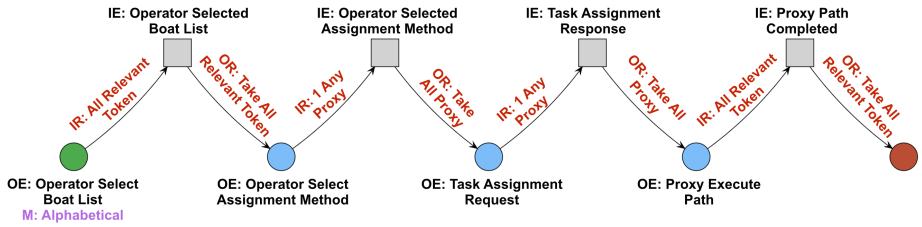


Fig. 1 SAMI Petri Net "Cooperative Location Visit" plan (without the interrupts). The starting place is colored green and the end place is colored red

In contrast, **static** sub-missions are instantiated only once, when the parent plan is instantiated, and have an empty initial marking. They share their start place with the parent plan: tokens that enter the parent *sub-mission place* are also added to the start place of the sub-mission.
All sub-missions can return values and tokens as well as write to variables shared with their parent plan. When a token(s) enters an end place in a sub-mission, the sub-mission is marked as being "complete." Until then, transitions in the parent plan leaving the sub-mission place are prevented from firing (Algorithm 1). When a transition fires, the completion status of any sub-mission in an incoming place is reset (Algorithm 2). Sub-missions allow developers to reduce repeated creation of common sequences and increase readability of the plan.

Markup: Each event e has a set of markup (using the mapping $MK = E \rightarrow \{mk_1, mk_2, ..., mk_n\}$, which maps each event to a set of markup). Markup are context clues associated to events which can provide several types of information: which GUI components and widgets are most appropriate for operator interaction, which set of priorities an agent service should consider when choosing from multiple algorithms, and which level of mixed-initiative autonomy to employ in making decisions.

Markups are an addition to the CPN model we consider here [10], which can be exploited to support situational awareness and mixed initiative control, making the model more flexible. We discuss markups here for completeness, however we do not use this concept in our empirical analysis nor in the definition of the interrupt mechanism that is the main focus of this paper.

Each *markup* $m \in MK(e)$ has a number of options and variables that the SPN developer must specify. GUI components and agent services correspondingly indicate which markup options they support, allowing the most appropriate ones to be retrieved automatically at run-time.

For example, the "relevant proxy" markup indicates to the GUI that the locational data of certain proxies should be displayed to the operator in addition to any other information contained within the event. Settings include the proxy selection criteria (the event's relevant proxies or all proxies) and which data to visualize (including pose, current path, future paths, and past paths). The "mixed initiative trigger" markup is used to indicate when system autonomy should make a decision and if the operator should be informed. Options range from never using system autonomy, using autonomy after a timer expires, or using autonomy immediately without consulting the operator.

The main components of an SPN are illustrated in a sample plan in Figure 1. When a plan is selected to run, an initial marking is applied to the plan's start place, $p_S \in P$ (the leftmost place, colored green). When a token enters an end place, $P_E \subset P, p_S \notin P_E$, the

plan terminates (the rightmost place, colored red). The initial marking is a generic token and a proxy token for each boat, which triggers Algorithm 3 when applied to p_S.

Operator Select Robot List is triggered asking the operator to select the boats that will participate in the plan from the list of corresponding proxy tokens it received. When the operator performs this action, an *Operator Selected Boat List* input event will be generated and matched to its transition in the SPN. Its received status is set to true and Algorithm 1 will be called. The transition will be enabled and fired via Algorithm 2, taking the *relevant tokens* (i.e., the tokens corresponding to the selected boat proxies) to the next place. The plan progresses in a similar way until the tokens reach the last place (i.e., all selected boats have completed their path).

When this happens the plan reaches the end place and is no longer active.

To illustrate how the concept of color is used for modelling a multi robot team in SPN, two consecutive markings of the CLV plan execution are shown in figures 2(a) and 2(b). The figures display the same SPN reported in Figure 1. These markings illustrate how the colored tokens (related to different boats) progress through the SPN. Figure 2(c) reports the state of the SPN where all proxy tokens are inside the *Proxy Execute Path* place. In this state of the SPN the related output event instructs the three platforms to execute the their path (shown in the rightmost image). The path that each platform must execute is specified by the task assignment algorithm which was selected by the operator in the preceding place (*Task Assignment Request*). In contrast to a plan specified with a non-colored PN, a single SPN defines the entire team plan, instead of representing one PN for each robotic platform.

Algorithm 1 Checks if a transition should be enabled

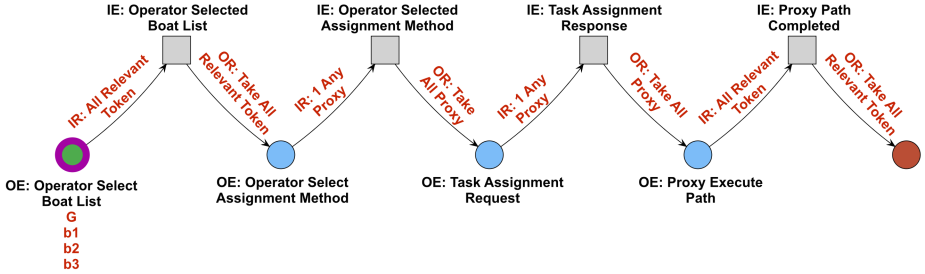
```
1: procedure CHECK TRANSITION
2:     for ie ∈ E_I(t) do                          ▷ Check that all input events have been received
3:         if ie.received == false then return false
4:         end if
5:     end for
6:     for if ∈ t.inEdges do          ▷ Check that all incoming edge's in requirements have been satisfied
7:         for ir ∈ R(if) do
8:             if ir.satisfied == false then return false
9:             end if
10:        end for
11:        p = if.start
12:        for sm ∈ SM(p) do      ▷ Check that any sub-missions on an incoming place are at a goal state
13:            if sm.complete == false then return false
14:            end if
15:        end for
16:    end for
17: return true
18: end procedure
```

3.2 Assisted plan designed and analysis for SAMI Petri Nets

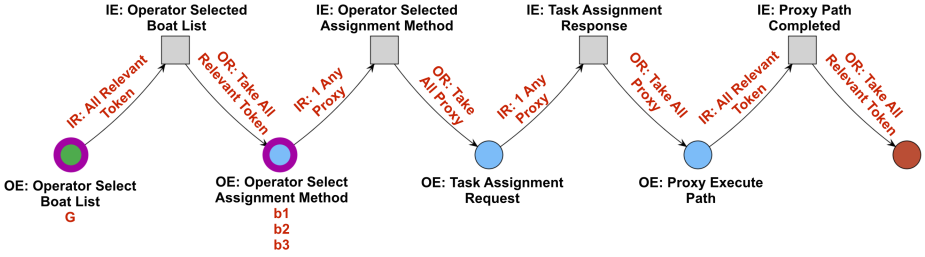
In order to assist the SPN developer, we created an intelligent plan editing tool. The editor was designed with two potential limitations of the plan language in mind: overwhelming visual clutter and developer errors resulting in an invalid SPN or unexpected run-time behavior. The editor contains different visualization modes which selectively hide and compress sections of nets based on different tasks the developer may be performing. “Assistant agents”

Algorithm 2 Fires an enabled transition

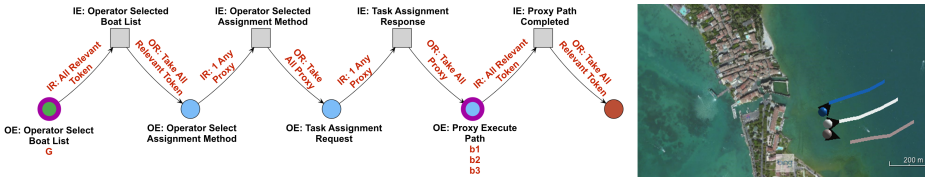
```
1: procedure FIRE TRANSITION
2:     t ∈ T                                              ▷ t is the transition we are executing
3:     TK_A = ∅          ▷ TK_A is a map associating tokens to add to outgoing places (initially empty)
4:     TK_R = ∅          ▷ TK_R is a map associating tokens to remove to incoming places (initially empty)
5:     for of ∈ t.outEdges do                             ▷ Fill in TK_A and TK_R
6:         for or ∈ R(of) do
7:             for p ∈ t.outPlaces do
8:                 TK_A.put(p, getTokensToAdd(or))
9:             end for
10:            for p ∈ t.inPlaces do
11:                TK_R.put(p, getTokensToRemove(or))
12:            end for
13:        end for
14:    end for
15:    for p ∈ t.outPlaces do
16:        enterPlace(p, TK_A(p))
17:    end for
18:    for p ∈ t.inPlaces do
19:        leavePlace(p, TK_R(p))
20:    end for
21:    for p ∈ t.inPlaces do          ▷ Reset completion status of all sub-missions on incoming places
22:        for sm ∈ SM(p) do
23:            sm.complete = false
24:        end for
25:    end for
26:    for ie ∈ E_I(t) do                  ▷ Reset receipt status of all input events on the transition
27:        ie.received = false
28:    end for
29:    T_check = ∅ ▷ T_check is a list of transitions we could have affected and should now check (initially
   empty)
30:    for p ∈ t.outPlaces do                             ▷ Fill in T_check
31:        for t2 ∈ p.outTransitions do
32:            if t2 ∉ T_check then
33:                t2 → T_check
34:            end if
35:        end for
36:    end for
37:    for p ∈ t.inPlaces do
38:        for t2 ∈ p.outTransitions do
39:            if t2 ∉ T_check then
40:                t2 → T_check
41:            end if
42:        end for
43:    end for
44:    for t2 ∈ T_check do
45:        if checkTransition(t2) == true then
46:            fireTransition(t2)
47:        end if
48:    end for
49: end procedure
```



(a) Initial marking of the CLV plan, with 3 tokens (associated to boats) and 1 generic token (a generic token is always included in the initial SPN marking to start the plan).



(b) This marking shows the state of the SPN after the operator selected the boats for executing the mission. At this point the 3 tokens representing the boats are moved to the next place in the SPN.



(c) This marking shows the state of the SPN when the proxy tokens are inside the *Proxy Execute Path* place. At this point each boat will execute its related path based on the task assignment algorithm which was selected by the operator in the previous place *Task Assignment Request*. The paths for the three boats are reported in the rightmost image

Fig. 2 SAMI Petri Net showing a partial execution of the "CLV" plan (without the interrupts)

Algorithm 3 Handles tokens entering a place

```
1:  procedure ENTERPLACE
2:      TK = {tk_1, tk_2, ..., tk_n}        ▷ TK is a list of tokens being added to the place
3:      for oe ∈ E_O(p) do
4:          processEvent(oe, TK)
5:      end for
6:      for sm ∈ SM(p) do
7:          beginSubMission(sm, TK)
8:      end for
9:      if p ∈ P_E then
10:         finishPlan(p, TK)
11:     end if
12: end procedure
```

check for violations of SPN rules and flag errors, such as incomplete graphs and unlabeled start/end places, and warnings, such as suspicious edge labels.

In addition to these checks that verify syntactic properties of the SPN, we can consider typical properties for PN and CPN such as the liveness and home properties [10].

In more detail, as discussed in [32], some properties are particularly interesting for plan monitoring frameworks. Specifically, in [32] the authors state that a PNP must be minimal (i.e., all transitions can be fired at least once), effective (i.e., the goal marking is a home state), and safe (i.e., the Petri Net is 1-bounded).

For what concerns our framework, SPNs that specify valid team plans should also be minimal and effective. Specifically, SPN should be minimal as they should not contain transitions that will never fire. Moreover, SPNs should be effective, because they encode team plans and as such they explicitly have a goal marking that must be reachable from all possible markings of the SPN (i.e., the goal marking should be a home state). However, the safety property does not apply to our framework. PNP tokens define execution threads for atomic actions, hence there should not be two tokens in the same place.

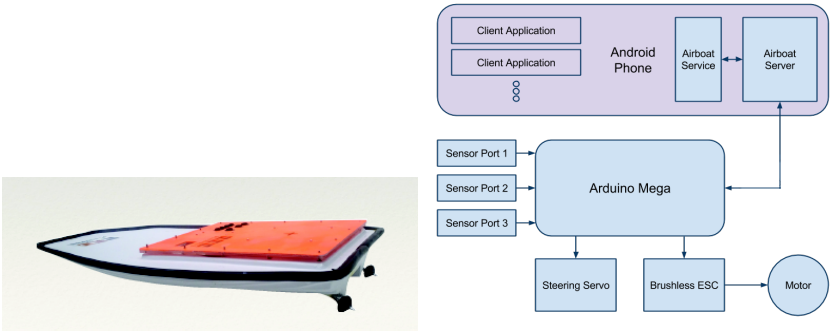
In contrast, an SPN place could have many tokens, and the tokens are not necessarily of different colors; several tokens of the same color set (for example, proxy tokens for Boat A) could exist in the same place simultaneously. A motivation for this would be knowing how many times a particular proxy has triggered a contingency behavior by counting the number of proxy tokens for that proxy which are in a particular place.

The above described properties (i.e., an SPN being minimal and effective) can be checked with standard reachability analysis performed on CPN. However, this requires to transform SPN plans to standard CPN (e.g., by removing output events from places and by associating them to new transitions, as mentioned above). We performed this analysis on the plans we consider here using CPN Tool [10] and our analysis reports that all plans we consider are both effective and minimal. Nonetheless, this does not imply that all SPN plans can be directly translated to an equivalent CPN and analysed using CPN Tools. This would require further investigations which fall outside the scope of the current contribution.

3.3 The cooperative robotic watercraft system

Figure 3(a) shows a differential drive propeller boat. In addition to a battery based propulsion mechanism, each boat is equipped with an Android OS smartphone, custom electronics board, and sensor payload. The Android smartphone provides communication, either through a wireless local area network or 3G cellular network, GPS, compass, and multi-core processor. An optional prism can be mounted to the transparent lid of the waterproof electronics bay to use the phone's camera for stationary obstacle avoidance and imaging. The Arduino Mega based electronics board receives commands from the Android phone over USB OTG and interfaces with the propulsion mechanism and sensor payload, as shown in Figure 3(b). The electronics board supports a wide variety of devices including acoustic doppler current profilers and sensors that measure electroconductivity, temperature, dissolved oxygen, and pH level. All sensor data is logged with time and location.

The robot team is controlled from a nearby base station via a high power wireless antenna or remotely using 3G connectivity. The operator uses a SAMI compatible GUI to instantiate SPN plans, monitor their execution, and provide input as necessary. In this case, compatibility means the GUI contains a library of UI components listing which data classes and SAMI markup they support, allowing a custom "interaction panel" to be constructed for each event requiring operator input.



(a) The Lutra-T Platypus robotic platform (b) System architecture

Fig. 3 A Platypus robotic platform with twin propeller and a diagram of the system architecture

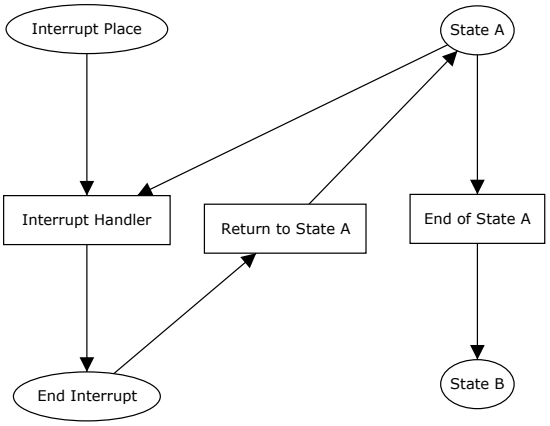


Fig. 4 Interrupt implementation with Petri Net.

4 The Interrupt Mechanism

In this section, we describe the basic ideas of the interrupt mechanism for Petri Nets and how we realize such mechanism in SAMI. Then we discuss an exemplar multi-agent plan that makes use of such interrupt mechanism (i.e., the Cooperative Location Visit plan).

4.1 Modeling the interrupt Mechanism in Petri Net

The Petri Net paradigm does not offer a special construct to implement interrupts, but it is possible to replicate the behavior of an interrupt through a specific sequence of places and transitions [6].

Figure 4 reports an example of an interrupt realized in the Petri Net framework. Essentially, the normal execution flow can be interrupted when the system is in *state A*. The interrupt can be triggered by the human operator simply placing a token in the *Interrupt Place*. This will enable the *Interrupt Handler* transition, hence changing the execution flow of the plan. If the *Interrupt Handler* transition fires, the system will place a token in the

End Interrupt place, and, when the execution of such behavior is completed (i.e., when the *Return to State A* transition fires), the system resumes the normal execution by placing a token back to the *State A* place. Notice that during the execution of the interrupt behavior, the transition *End of State A* is not enabled, therefore the flow of execution can not progress to *State B* until the interrupt handler behavior is completed.

4.2 Modeling the Interrupt mechanism in SAMI

Following the interrupt implementation idea described in Figure 4, we use three key elements to model the interrupt mechanism in the *SAMI* framework: i) a place (called *Interrupt place*) ii) a transition that starts the interrupt handling procedure (*Start interrupt transition*) and, iii) a transition that determines the end of the interrupt procedure (*End interrupt transition*). Now, consider a generic plan that we represent with a *Source place*, indicating the state of the system that could receive an interrupt, a transition, indicating some part of a plan, and a *Destination place*, indicating the state of the system that should be reached when the interrupt handling procedures terminates (consider that the source and destination places could be the same).

Figures 5(a) and 5(b) show the CPN structures we propose to add interrupts to. We consider two types of interrupts: a *proxy* interrupt (see Figure 5(a)) and a *general* interrupt (see Figure 5(b)). As the figures show, the structure to realize these two types of interrupts is the same; however, the events attached to the places/transitions and the requirements on the edges of the net are different. In both structures, the *Start interrupt transition* and the *End interrupt transition* are connected by a *Sub-mission interrupt place* which represents a sub-mission that models the appropriate interrupt handling behavior. After the execution of the sub-mission all the tokens returned by the sub-mission (i.e., the tokens which completed the sub-mission) move to the destination place of the interrupt, and restore the normal behavior of the plan. Below we describe these two interrupt types in more detail.

Proxy Interrupt The *proxy* interrupt relates to a specific subset of the platforms, and affects the execution flow of those platforms only (while the others continue the normal execution of the plan). This type of interrupt typically represents a procedure that should be activated in response to some proxy-level events, e.g., the battery of a boat reaches a critical level and the boat should stop the current plan to go to a recharge area.

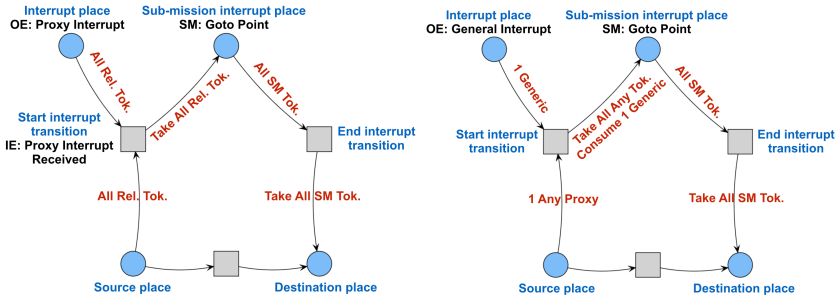
In particular, the interrupt place generates a *Proxy Interrupt*, which is an output event[3]. The *Proxy Interrupt Received* input event encapsulates the information regarding which proxies should be involved in the event. Such information is used by the *Start interrupt transition* to take only the relevant tokens from the *Source place* and move them to the *Sub-mission interrupt place*. Consequently, only the tokens specified by *Proxy Interrupt Received* will stop their current plan to execute the interrupt sub-mission. Such relevant tokens are selected with a plan specific procedure, and this often requires a user interaction (i.e., the user directly selects which platforms should execute the interrupt sub-mission).

General Interrupt The *general* interrupt is a team-level interrupt that is not specific to a particular platform. The *general* interrupt represents a situation where all robotic-boats should perform a particular procedure, e.g., stop all current plans and go to a safe position as a manned boat is approaching.

[3] Recall from Section 3.1 that output events are associated to places and contain commands or requests for other modules. Input events are associated to transitions and encapsulate information that should be consumed by the module that receives such event

In contrast to the *proxy* interrupt, the *general* interrupt will remove all tokens present in the *Source place* and transfer them to the sub-mission. Hence, the event generated by the *Interrupt place* is a different output event, named *General Interrupt*. Such event is generated to trigger the interrupt mechanism but does not contain any specific information regarding the relevant proxies (as all proxies are relevant in this case). Consequently, the *Start interrupt transition* requires a *generic* token (and not a proxy token) and it will transfer all the *proxy* tokens from the *Source place* to the *Sub-misison interrupt place*. Note that, unlike a proxy interrupt, a general interrupt has no input event on the start interrupt transition, as it always moves all tokens and thus does not require any additional information. A general interrupt is essentially a compact way of representing an interrupt for all proxies. Such compact representation is crucial for team level plans that must be designed and monitored by human operators.

The interrupt parts of the SPN are not logically different from non-interrupt parts. Hence, since SPN supports sub-missions, we can also have nested interrupts.



(a) Proxy Interrupt (b) General Interrupt

Fig. 5 Types of interrupt implemented in the SPN framework.

4.3 Using interrupt in multi-agent plans

Here we provide an exemplar multi-agent plan, discussing the possible use of both interrupt types described above. In particular we consider a Cooperative Location Visit (CLV) plan where the operator selects a group of boats to visit a set of locations to perform point measuring tasks. The boats should navigate to each location and acquire a specific measure (e.g., pH level, oxygen level, temperature). In this work, we assume that each boat is equipped with the same sensors, hence visiting the same location with different boats does not provide more information and should be avoided, in contrast, each boat can visit several locations (i.e., executing a path that goes through all such locations in sequence). The system offers various techniques to assign boats to locations and in this work we used a method which is based on Sequential Single Item auctions [25]. The method assigns locations to boats sequentially, and for each location the system selects the boat that can provide the lowest path cost. Such path cost is computed as the minimum path cost that the boat can

achieve when inserting the current location in the set of locations that are already assigned to such boat[4].

The CLV plan is reported in Figure 6. In such a plan, the general interrupt handles a situation where the user decides to temporarily stop the current plan of all the boats to avoid a dangerous situation, i.e., a manned boat that enters the area where the boats are operating. The general interrupt starts from the *Proxy Execute Path* place and goes back to the same place. When the interrupt triggers, all the tokens present in the *Proxy Execute Path* place are transferred to the sub-mission place. This token transfer requires the presence of at least one *Proxy* token in the *Proxy Execute Path* place and is performed by using the *take* action (see Section 3.1) on all Proxy tokens that are present in such place. As mentioned in Section 3.1 the *take* action will remove the specified tokens from the incoming place and will add them to the outgoing place, which in this case is the *Assemble* sub-mission (SPN not shown). Hence the effect of this token transfer is that all proxies will stop executing the current action and will start the *Assemble* sub-mission. Such sub-mission, sends all the boats to a specific safe assemble position and then waits for operator input to end the plan, allowing the parent plan to continue. When the operator decides that the dangerous situation is over, the *End general interrupt* transition fires and boats are sent back to the *Proxy Execute Path* place, where they resume executing the plan, maintaining their previous location assignments. This token transfer is triggered by the *End general interrupt* event and is performed with the *take* action on all sub-mission tokens. The sub-mission tokens are the set of tokens which reached the end place in the sub-mission; in this case, these are the proxy tokens for the boats which were station keeping to avoid the danger. The *take* action means that the proxy tokens will be removed from the *Start sub-mission* place and added to the *Proxy Execute Path* place.

In contrast, the proxy interrupt allows the operator to stop the execution of a selected subset of the boats without interfering with the plan execution of the other boats. This is useful when the human operator should handle an event that influences the behavior of a specific group of boats, i.e., a boat that reaches a critically low battery level. The proxy interrupt moves the set of selected proxies to the sub-mission place while the others will continue their execution. In our exemplar plan, the sub-mission associated with the interrupt, *Recharge*, pauses the current plans of the provided proxies and sends them to a recharge station, where batteries are replaced with fully charged ones. The sub-mission then ends, allowing *End proxy interrupt* to fire, which moves the proxies back to *Proxy Execute Path* where they resume visiting locations. Similar to the general interrupt, we use the *take* action to transfer tokens from the *Proxy Execute Path* place to the *Recharge* sub-mission and then the *take* action to transfer them back. However, in this case we take from the *Proxy Execute Path* place only the *Relevant* tokens, i.e. the tokens associated to proxies that must be recharged. As mentioned in Section 4.2, the information regarding which tokens are relevant is specified by the input event *Proxy Interrupt Received* associated to the *Start Proxy Interrupt* transition.

Depending on the specific plan and on the desired behavior for the interrupt sub-mission, we might need to insert extra elements into the basic plan. An example of this is the plan to handle the traverse dangerous area event, shown in Figure 7 and discussed in detail in Section 5.

By combining the team-level and proxy-level interrupts our approach provides a powerful and general model to allow sophisticated interactions between the human operators and

[4] Since computing the minimum path cost given a sequence of visit locations is in general NP-Hard here we use a simple nearest neighbor heuristic: the path is built incrementally by always selecting the next location as the one that is closest to the current location. At the beginning the current location is the boat position.

the robotic system. As the empirical evaluation shows, this results in a significant performance gain for the system.

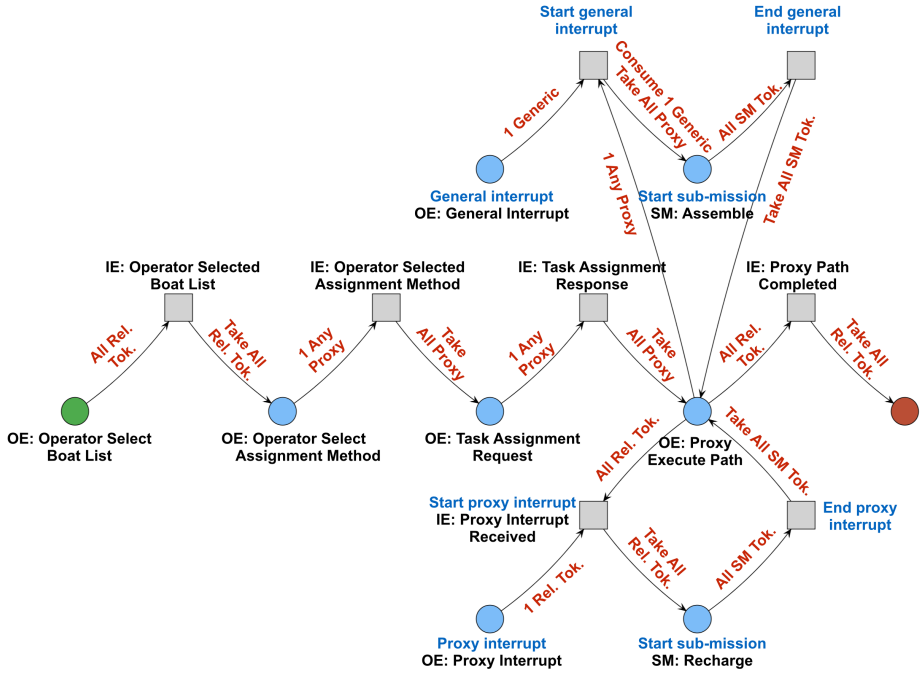


Fig. 6 The Cooperative Location Visit plan specified in the SPN framework, with both general and proxy interrupts.

5 Empirical Results

In this section we present a quantitative evaluation of our approach to team plan monitoring in the CRW domain. We first describe our empirical methodology, then we present and discuss the results we obtained.

5.1 Empirical Methodology

The main goals of the empirical evaluation are: i) to validate the applicability of the interrupt mechanism to team-level plans that represent realistic use cases, ii) to evaluate the gain achieved by such a mechanism, in terms of task specific performance as well as operator load, with respect to aborting the plan when an incident arises.

As a first step, we consider two versions of the CLV plan discussed in Section 4.3: the "interrupt" version which encodes interrupts within the plan (reported in Figure 6) and the "standard" version without any interrupts (reported in Figure 1). Next, we define three possible incidents: i) *general alarm*, ii) *temporary boat pull-out* and iii) *traverse a dangerous*

area. We then simulate the execution of both versions of the CLV plan for each incident, measuring indicators of task specific performance and operator work load. When we execute the standard plan and one of the incidents takes place, the human operator must abort the entire plan's execution, execute the plan that can resolve the incident, and then start a new instance the original plan once the resolution plan has finished.

In more detail, the incidents and the co-related team behaviors have been defined as follows:

General alarm represents a danger that may significantly interfere with the plan execution of all the boats. An example of this could be a manned boat that enters the operative areas of the robotic boats. If this happens the human operator should signal to all the platforms that all plans should be suspended to avoid collisions. When the manned boat leaves the scene the human operator can then instruct the boats to recover the execution of their plans (i.e., execute the remaining tasks). This situation can be handled with a general interrupt as all the boats will have to execute the same specific sub-mission (i.e., reach a safe position) before recovering their plans. In our empirical evaluation we simulate the occurrences of several general alarm incidents while a CLV plan is running. In particular, we fix the number of incidents to happen and distribute them randomly during the plan execution.

Temporary boat pull out represents an incident that interferes with a specific subset of robotic platforms and that will not directly hinder the plan execution for the rest of the team. An example of this could be the need to recharge the battery for one robotic boat. Specifically, we simulate a discharge process for the boats, where the battery level is reduced based on distance traveled. The discharge process includes a random element that increases or decreases the units of battery consumed to simulate possible not-modeled situations (such as currents) that impact the amount of energy required to traverse a given distance. In more detail, if we indicate with $b_i(t)$ the level of battery at time t for boat i, we have that $b_i(t+\tau) = b_i(t) - Kd_i(\tau)(1+R)$, where τ is a positive value that represents a time interval, $d_i(\tau)$ represents the distance (in meters) traveled by boat i in the time interval τ, K is a constant that expresses the units of battery required to travel one meter, and $R \sim U(-0.1, 0.1)$ is a random variable drawn from a uniform probability distribution.

Traverse dangerous area represents an incident where several boats must traverse an area that is problematic for navigation. For example consider a scenario where a part of the intervention area is cluttered with objects (e.g., vegetation, pieces of wood, etc.) or presents strong currents. In this situation, we require a human operator to constantly monitor the operation of the platforms to be able to promptly intervene (i.e., teleoperating the boats) if necessary. Since it is impossible for a single operator to effectively monitor and teleoperate multiple boats at the same time, a key element for this plan is to synchronize the execution of the boats making sure that only one boat is actively navigating in the dangerous area, while other boats that might need to traverse the same area will wait for the availability of the human operator.

In the standard plan without interrupts, the operator should abort the plan, which means all boats should stop what they were doing. The human operator can then monitor the boats inside the area sequentially. Boats outside the area will be stopped until there is only one boat inside the area, then the plan will resume which means that all remaining tasks will be reassigned. If we execute the plan with the interrupt mechanism, the operator can choose to monitor one platform while all other boats that are inside the area will be stopped until the

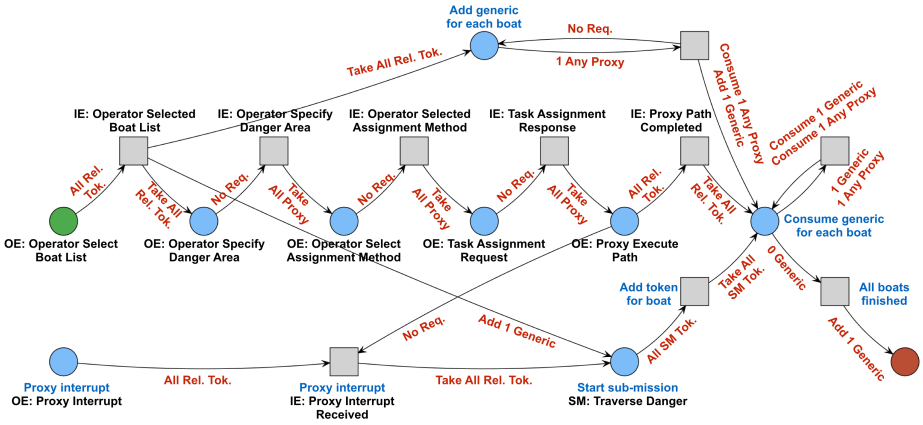
human operator becomes available for close monitoring. Meanwhile other boats outside the area will continue their paths.

Figure 7 reports the CLV plan with a proxy interrupt to handle the traverse dangerous area incident. Specifically we report the parent plan in Figure 7(a) and the traverse dangerous area sub-mission plan in Figure 7(b). Notice that in the parent plan (Figure 7(a)) proxy tokens can follow two different branches to reach the end place of the plan, depending on whether they enter a dangerous area or not. Since in this case the plan should terminate only when all boats have finished their paths (i.e., boats that never entered the dangerous area in addition to boats that did), as mentioned in Section 4.3 we must insert extra transitions and places to make sure that the plan will terminate only when all boats have visited their assigned locations. This is the role of the place labeled *Consume generic for each boat*. In more detail, this place will accumulate one generic token for each platform that is selected by the operator (this is done through the loop in the upper part of the plan). Then when the proxy tokens representing the platforms reach this place, such generic tokens will be consumed (this is done through the loop in the left part of the plan). The plan will then terminate only when all such generic tokens have been removed. This is done through the last transition (*All boats finished*) which effectively represents an inhibitor arc (it will fire when there are no tokens in the preceding place).[5] Notice that the structure of the interrupt is the same as the one reported in Figure 5(a), i.e., we have a place that enables the interrupt associated to the output event *Proxy Interrupt* and a start transition for the interrupt (associated to the input event *Proxy Interrupt Received*) that moves only relevant proxy tokens (i.e., only boats that are inside the dangerous area) to the interrupt sub-mission.

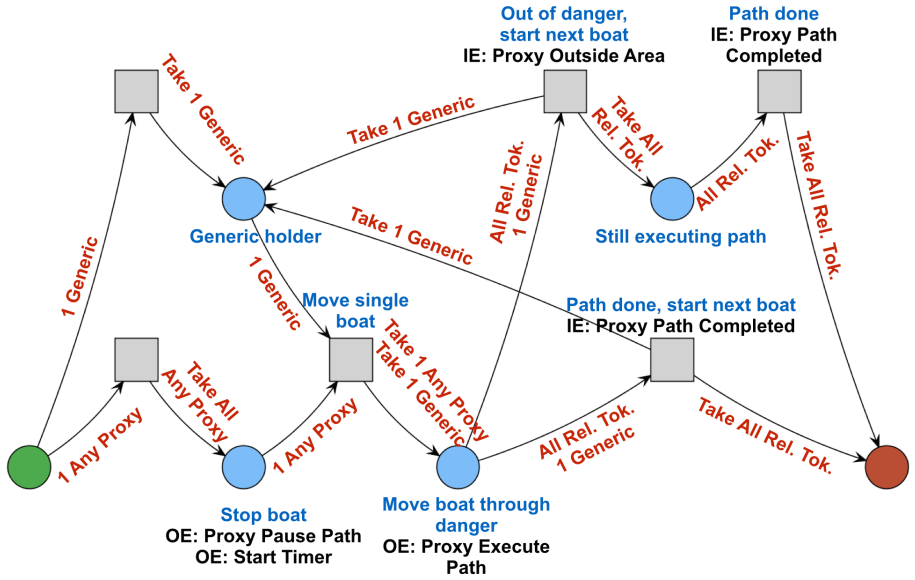
The "Traverse Dangerous Area" sub-mission reported in Figure 7(b) is used as static sub-mission (see Section 3.1) in the *Start sub-mission* place. Thus, when the transition holding the input event *Operator Selected Boat List* in the CLV plan fires, the generic token added to the *Start sub-mission* place is also added to the start place of the single instance of the sub-mission. In the sub-mission, the generic token will then be moved to the *Generic holder* place. This place is crucial to synchronize the behaviors of the platforms: if a proxy token enters the sub-mission, the corresponding boat will be stopped and it will not be allowed to execute the remaining path unless there is a token in the *Generic holder* place. Since the transition *Move single boat* takes that generic token, only one boat at a time will be allowed to execute the path inside the dangerous area. The next boat will start the path execution only when the boat currently traversing the dangerous area has completed its path (i.e., when the *Path done, start next boat* transition fires) or it is out of the dangerous area (i.e., when the *Out of danger, start next boat* transition fires). That is because both these transitions put a generic token back in the *Generic Holder* place. Note that these two transitions are mutually exclusive, so it is not possible for both of them to trigger, which would result in two generic tokens being place in *Generic holder*. Overall this plan represents a complex team oriented plan that requires a sophisticated synchronization between the boats, however the interrupt mechanism and the use of advanced features of the SPN framework (such as the static sub-mission) allows us to realize such a plan in a fairly compact structure.

Execution model for the system In our experiments we adopt the following execution model for the system: when we execute the interrupt version of a plan, with interrupt mechanisms in place, we assume that whenever an incident requiring intervention arises, the operator will trigger the corresponding interrupt. For example, when we execute the CLV

[5] While in our case the number of proxy / generic tokens is always finite, we might not know this number before the plan starts. Hence we use the inhibitor arc to check whether a place is empty.



(a) The parent plan



(b) The (static) sub-mission for the traverse dangerous area

Fig. 7 CLV plan with the interrupt for traverse dangerous area

plan and the battery level of a boat reaches a critical level, in our simulation the corresponding proxy interrupt will always be triggered and the correct boat will be selected. In other words, we assume the human operator will always do the correct actions that the framework offers to respond to an incident. This is because our intent here is to evaluate the interrupt mechanism and not the human interface. As mentioned in the intro, a proper evaluation of the human interface falls outside the scope of this contribution.

When we execute the standard version of the plan, which lacks interrupts, we assume that the human operator will abort the current plan, start a new plan(s) to handle the incident and, finally, when the incident has been resolved (e.g., a low battery has been swapped), they will start a new instance of the original plan to complete its objectives. Note that, when

the operator starts the new instance of the original plan, all required information must be re-inserted, such as the locations to visit. In our experiments, we assume the operator can keep track of which locations have been visited and re-start the plan only with the locations yet to be visited (reducing the number of interactions in favor of the standard approach). Moreover, we assume that the operator will start the new instance of the original plan only after the plan(s) used to resolve the incident has been completed. For incidents which do not affect the entire team (e.g., a boat with a low battery requiring a pull out and a subset of the team needing to traverse a dangerous area), this means that some of the team will remain idle when the original plan is aborted, even though they are not involved in the incident. We further investigate this with a second set of plans for the temporary boat pull out scenario. In these "reassignment strategy" versions of the standard and interrupt plans, when a boat leaves to swap its battery, the rest of the team continues with its tasks. Furthermore, we reassign the locations that boat was responsible for to the other members of the team. When the battery swap is finished, we reassign all tasks that must still be accomplished to all boats. Note that, while the commands sent to the boat team are identical for the standard and interrupt versions of the plan for the reassignment strategy, the actual SPNs and the way the operator interacts with them to respond to the low battery incident are different.

Metrics The metrics we extract from the simulation combine task dependent metrics and metrics to evaluate the operator load. Specifically, the task dependent metric is the time to complete a plan while the load metric is the number of user actions required to start/abort the plan, trigger the interrupt, provide information to the boats (e.g., the locations to visit). In our experiments such interactions always take the form of a click (on a map or on a button), hence we measure the number of clicks that the operator performs. Since the main goal of the empirical evaluation is to compare the use of the interact mechanism with the standard execution model, we compute and report the percentage gain of the interrupt mechanism for both metrics. In particular, we compute $\frac{(v_{Std}-v_{Int})}{\max\{v_{Int},v_{Std}\}} * 100$, where v_{Std} is the value of the metric obtained with the standard execution model and v_{Int} is the value of the metric obtained with the interrupt mechanism. Since for both metrics the lower the better, a positive value indicates superior performance of the interrupt mechanism over the standard execution model.

In all the following experiments, the interrupt mechanism does not provide additional domain knowledge with respect to the standard plan execution. In particular, the recovery procedure for handling the incidents is the same when using interrupt and when aborting plans. Overall, our goal here is to provide a domain-independent interrupt mechanism, for the SPN plan specification language, which can select the most appropriate domain-dependent recovery procedure when an incident happens. Moreover, we aim at doing this in a smooth way (i.e., without stopping and restarting the plan that is currently running). While one could potentially devise a different domain-specific mechanism to select the most suitable recovery procedure this would defeat the purpose of using a general plan specification language such as SPN.

In this perspective, the gain we obtain is due to the presence of the interrupt mechanism that smoothly changes plan execution instead of aborting and restarting. Consequently, in most situations the interrupt mechanism will require fewer interactions, because we need at least the same number of user interactions to stop and re-start the plan compared to interrupting it. However, for completion time there might be situations where having the interrupt mechanism does not help (e.g., see results for Table 1).

In the next section we report and discuss the results obtained with our empirical evaluation.

5.2 Quantitative results in simulation environment

Configurations	Std	Int.	% Gain (Interrupt vs Standard)	
#boat,#loc.,r.t.	#rec.	#rec.	Total Time	# interactions
3, 20, 10	6	6	6.3%	73%
5, 20, 10	5	5	23% [± 0.5]	68%
3, 20, 20	6	6	26% [± 2.5]	72% [± 0.8]
5, 20, 20	5	5	27% [± 6.6]	64% [± 3.7]
3, 30, 10	11	12	26% [± 1.2]	69% [± 9.5]
5, 30, 10	10	12	21%	75%
3, 30, 20	11	12	48% [± 0.8]	80% [± 0.1]
5, 30, 20	10	12	27% [± 2.9]	75% [± 0.5]

Table 1 Results for the CLV plan and boat pull out event. Each configuration specifies the number of boats, the number of locations, the time required to recharge the boat's battery (in seconds). The number of recharge (#rec) represents the number of times a boat required a recharge action for the standard execution (Std.) and for the plan with the interrupt (Int.)

Configurations	% Gain (Interrupt vs Standard)
#boat,#loc.,#alarms	# interactions
3, 20, 1	44% [± 0.6]
5, 20, 1	40% [± 1.4]
3, 20, 3	65% [± 0.6]
5, 20, 3	61% [± 1]
3, 30, 1	46% [± 0.3]
5, 30, 1	16% [± 1.9]
3, 30, 3	68% [± 0.23]
5, 30, 3	66% [± 0.4]

Table 2 Results for the CLV plan and the general alarm event. Each configuration specifies the number of boats, the number of locations and the number of alarms.

Configurations	% Gain (Interrupt vs Standard)	
#boat,#loc.#boats inside area	Total Time	# interactions
3, 20, 2	5.2% [± 2.9]	40.2% [± 2.16]
5, 20, 2	6.9% [± 2.2]	39.1% [± 0.5]
3, 20, 3	10.4% [± 1.7]	42.5% [± 0.6]
5, 20, 3	9.8% [± 1.8]	42.9% [± 1.1]
3, 30, 2	(4.3% [± 2])	45.3% [± 1.6]
5, 30, 2	9.9% [± 2.4]	43.6% [± 1.3]
3, 30, 3	5.4% [± 1.3]	43.6% [± 0.5]
5, 30, 3	15.9% [± 1.7]	44.4% [± 0.5]

Table 3 Results for the CLV plan and enter dangerous area event. Each configuration specifies the number of boats, the number of locations and the number of boats that are inside the dangerous area at the same time (the value between parenthesis is not statistically significant according to a t-test with $\alpha = 0.05$, all others are).

	Simple Strategy		Reassignment Strategy
Configurations	% Gain (Interrupt vs Standard)		% Gain (Interrupt vs Standard)
#boat,#loc.,#rec,r.t.	Total Time	#interactions	# interactions
3, 20, 3,10	11%	65%	80%
5, 20, 3,10	16%	65.4%	81%
3, 20, 3, 20	14.8%	64%	79.6%
5, 20, 3, 20	13.4%	63.4%	78.7%
3, 30, 5,10	13%	75.6%	86%
5, 30, 5,10	17%	73%	85%
3, 30, 5, 20	16.8%	76%	86%
5, 30, 5, 20	11%	76.6%	83%

Table 4 Results for the CLV plan and boat pull out incident for the previous simple strategy (do not reassign tasks) and reassignment strategy. Each configuration specifies the number of boats, the number of locations, the time required to recharge the boat's battery (in seconds). The number of recharge (#rec) represents the number of times a boat required a recharge action which is assumed to be 3 for 20 locations and 5 for 30 locations in these experiments.

Table 1 reports results obtained for the CLV plan and the boat pull out incident. In particular, we consider a set of configurations, where each configuration is defined by three elements: i) the number of boats involved in the plan (3,5), ii) the number of locations to be visited (20,30) and iii) the time required to exchange a boat's battery expressed in seconds (10,20). For each configuration we executed 10 repetitions. We report the average values of the gain for both metrics and the standard error of the mean (shown in square brackets). In the tables, we report only the percentage gain for configurations that show a statistically significant difference between the values of the means[6].

As it is possible to see, for all configurations the plan with the interrupts achieves better performance both in terms of time to complete the plan as well as for the operator workload. In more detail, focusing on the time to complete the plan, we can see that the gain of the interrupt mechanism with respect to the standard mechanism increases when the recharge time increases, because in the standard execution model all plans must be aborted when a boat must recharge, while in the interrupt model the other boats can continue with their plan execution. As for the operator work load, the interrupt mechanism requires far fewer user actions than the standard plan. This is due to the fact that, in the standard execution model, the user must re-insert the locations that the boats must visit when the CLV plan is re-started. Notice that the number of recharge actions is higher when using the interrupts model. This is because the standard mechanism re-starts the whole plan each time a boat must be re-charged, consequently the remaining locations to be visited will be re-allocated among the currently available platforms. This provides solutions of higher quality for the allocation process (i.e., shorter paths), compared to the interrupt mechanism, which uses the same solution throughout the entire plan execution. Therefore, when using the interrupt mechanism boats might end up traveling more, and since the battery discharge process depends on the traveled distance, this results in more recharge actions. However, as results clearly show, this is compensated by a significant reduction in time to complete the plan and operator load.

Table 2 reports results achieved for the CLV plan and the general alarm incident. We considered the same number of boats and number of tasks, and we vary the number of alarm incidents that will appear during the plan (1,3). As before, we report the average values of the gain and the standard error of the mean.

[6] To check whether results are statistically significant we run a t-test with $\alpha = 0.05$.

Concerning the operator work load, these results confirm the superior performance of the approach that encodes interrupts in the plan. However, in this case, the difference in time to complete the plan does not show a statistical significance, consequently we do not report such values. This is because the procedure to handle the general alarm requires all boats to stop and wait until the original plan can be safely re-started. Hence, the actions that the boats perform when aborting a plan are very similar to the interrupt handling procedure. In all the simulations we do not consider the time required by a human operator to perform the click actions but we simply count the number of clicks. This is because a proper evaluation of such time would be highly dependent on the skills of the operator. However, in practice this time will not be negligible and would significantly increase the gain in favor of the interrupt mechanism.

Table 3 presents results for the CLV plan with the traverse dangerous area incident. Again we consider the same number of boats and tasks and we vary the number of boats that simultaneously enter the dangerous area during the plan (2,3). In this case, if a single boat is inside the dangerous area there is no need for interrupting the plan. This is because the plan monitoring framework allows the operator to override boat autonomy at any time, directly teleoperating a single platform without aborting the current plan. Hence, if a single boat is traversing the dangerous area the operator can focus his/her attention on such a boat without changing the behaviors of the other platforms. However, if more than one platform are traversing the dangerous area at the same time, the plan must be changed to stop all boats inside the area so to focus operator attention on a single one. Hence, in our experiments, we consider only situations where at least two boats are simultaneously inside the dangerous area.

Results shows that also for this type of incident the interrupt mechanism provides an important gain (about 40%) in operator load and that such a gain does not vary significantly across the considered configurations. This is reasonable as the number of interactions that the operator must perform does not depend on number of boats and only marginally on the number of visit locations: in the standard version of the plan the operator will have to re-insert a higher number of locations when re-starting the plan, this is confirmed by a small increase in the gain when there are 30 locations to visit. As for completion time, the gain is less significant and there is no clear trend with respect to the configurations we considered. In fact, in this case, the gain depends on how tasks are placed with respect to the dangerous area. In any case, the use of our interrupt mechanism is providing a positive gain in all the configurations we considered.

Table 4 shows the results obtained for the CLV plan and the boat pull out incident using two different incident handling strategies, as described above. The goal of this set of experiments is to assess the flexibility of our interrupt mechanism and investigate whether the efficiency of the interrupt structure is dependent on the use of particular sub-missions. We consider the set of configurations used in Table 1, but to better compare the two plans we now assume a fixed number of recharge incidents during the plan (i.e. 3 boat pull out incidents for 20 locations and 5 incidents for 30 locations). The first two columns present the results using the same handling strategy and plans as in Table 1, while the third column shows the results for number of interactions for the reassignment strategy version of the standard and interrupt plans[7]. As mentioned previously, in the reassignment strategy versions of the plans, whenever the boat pull out incident occurs, the related boat will go to the base station for recharging while the remaining tasks are reassigned to the other boats, which

[7] According to a t-test with $\alpha = 0.05$, the total time gain for the reassignment versions of the interrupt versus standard plan is not statistically significant, so we do not report such metric in the table.

continue visiting their assigned locations. When the boat is recharged, all the locations that must still be visited will be reassigned to all boats (including the recharged one).

Results show that the the total time gain for the reassignment sub-mission interrupt mechanism according to this metric is not significant. This is expected as in both the standard and interrupt plans, the boats are never idle, unlike the simple strategy version of the standard plan. However, the gain for number of interactions (clicks) significantly increases. This is because, when the interrupt mechanism is not used, the operator needs to reassign the tasks when the recharging boat goes to the base station and when it comes back. In contrast, when the interrupt mechanism is used everything is handled through the sub-mission hence there are fewer interactions. In summary, the key point is that the interrupt mechanism helps in terms of completion time and interactions, and it is a flexible and general approach that can be easily used with different sub-missions.

Finally, a video showing an exemplar execution of the CLV plan presented in Figure 6 is reported here[8]. The video shows that, when the general interrupt is triggered all the boats move through the interrupt branch and enter a recovery sub-mission that sends them all to a safe assembly location. When the alarm is over, the boats resume their previous plan. In contrast, when the proxy interrupt is triggered, the selected boat proceeds to the recharge area while the execution of the other boats progresses unchanged. When such boat completes the recharge plan, it returns to finish executing its previous plan.

The video shows how our mechanism allows the human operator to smoothly handle different types of interrupts during the execution phase of complex team-level plans.

5.3 Validation on robotic platforms

We validated the use of our approach for interacting with team oriented plans on real robotic platforms. Specifically, we performed several experiments where a single operator was in charge of monitoring and interacting with the operation of several boats (up to nine). Here we discuss a specific experiment where platforms are sequentially inserted into the water and, as they are added, they start to execute a *Connect and station keep plan* to maintain a specific predefined position. A video of an exemplar run for the connect and station keep experiment can be found here[9] while Figure 8 reports a picture of the same run.

The experiment has been conducted in a marine coastal area, and as it is possible to see, currents would make the boats float away when motors are shut down. To avoid this, when executing the connect and station keep plan, the boats will periodically turn on their motors to move toward a assembly positions specified when the plan is invoked (left of the screen). This is a crucial behavior to effectively deploy a large team of platforms. The video shows the boats executing the plan, the evolution of the CPN representation for this plan, and a few screen-shots of the graphical interface that the operator uses to monitor the plan.

In this experiment the interrupt mechanism is used to re-define the points where boats should perform station keeping. This is a general interrupt as all boats will change their behavior. The operator activates the interrupt at minute 1:50 of the video, and it is possible to see how all boats change their plan and perform the station keep behavior in a different position (center of the screen).[10] This behavior is used in field deployments when large speed boats approach the current station keeping location, risking a collision with the robots.

[8] `http://profs.sci.univr.it/~farinelli/videos/CLV.mp4`

[9] `https://youtu.be/l5Qhp1JSoNI`

[10] This video was accepted to the IJCAI 2015 video competition.

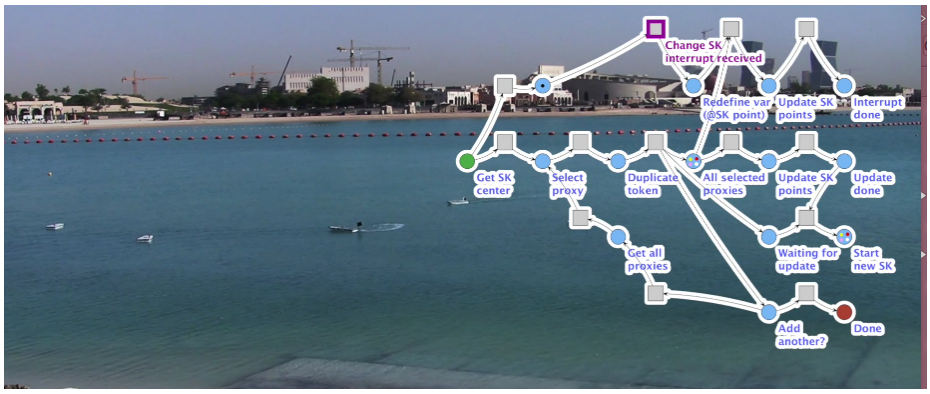


Fig. 8 A picture of the connect and station keep experiment. The image shows a subset of the platforms and the current state of the CPN representing the connect and station keep plan. The interrupt portion of the plan is visible in the top part of the picture and the current enabled transition (highlighted in the picture) is the one that starts the interrupt to change the position where boats should perform station keeping.

These experiments confirmed that our interrupt mechanism helps human operators to easily control the deployment of real robotic platforms.

6 Conclusions and future work

We consider the problem of handling human interrupts in team oriented plans. Team oriented plans are a key tool for allowing human operators to specify high level directives for teams of autonomous agents. However, in many scenarios an operator might need to interrupt the activities of individual team members to deal with particular situations (i.e., a danger that the team can not perceive). Previous to this work, after such an interruption the operator would usually need to restart the team plan manually to ensure its success.

Here, we proposed a mechanism that allows a range of interrupts to be handled smoothly, allowing the team to efficiently continue with its tasks after an operator intervention. In particular, we built on the SPN framework, which proposes the use of Colored Petri Nets for specifying team plans, and we defined two types of interrupts: a *proxy* interrupt that affects the execution flow of a subset of the platforms, and a *general* interrupt that specifies a particular recovery procedure for all the platforms.

We validated our approach considering an application of robotic watercraft. In more detail, we provided a quantitative evaluation of our interrupt mechanism by simulating the plan execution with and without the interrupts in a set of selected use cases. The empirical results show that, by combining the team-level and proxy-level interrupts, our mechanism provides a powerful and general model to allow sophisticated interactions between the human operators and team plans, resulting in a significant performance gain for the system. Moreover, we validated our approach on real platforms performing various experiments where a human operator should monitor and control the evaluation of several boats. Such experiments indicate that our mechanism can be of practical use in the actual deployment of robotic watercraft.

Many possible future directions stem from this work. A first interesting direction is to extend the current plan specification framework to perform the analysis described in Section 3.2 (e.g., reachability analysis) directly on the SPN. Moreover, we are currently working

on mechanisms to analyse whether the introduction of an interrupt makes an SPN effective even in face of possible action failures. These would be two important additions to assist the human operators in the design phase of the SPN plans.

Another interesting direction that touches upon similar issues is to evaluate how difficult is it for a human designer to learn and use the SPN plan specification framework and the associated interrupt mechanism. A possibility, is to perform a user study to evaluate whether human operators (possibly with different backgrounds and education) can design SPN plans that are both efficient and effective.

Acknowledgments

This work is partially funded by the Qatar National Research Fund NPRP grant 4-1330-1-213. This work is partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 689341. This work reflects only the authors' view and the EASME is not responsible for any use that may be made of the information it contains.

References

1. Bernard Berthomieu, Didier Lime, Olivier H Roux, and François Vernadat. Reachability problems and abstract state spaces for time petri nets with stopwatches. *Discrete Event Dynamic Systems*, 17(2):133–158, 2007.
2. J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(3):367–385, June 2003.
3. Philip R Cohen and Hector J Levesque. Teamwork. *Special Issue in cognitive Science and Artificial Intelligence*, pages 487–512, 1991.
4. John Collins, Corey Bilot, Maria Gini, and Bamshad Mobasher. Mixed-initiative decision support in agent-based automated contracting. In *Proceedings of the Fourth International Conference on Autonomous Agents*, AGENTS '00, pages 247–254, New York, NY, USA, 2000. ACM.
5. Hugo Costelha and Pedro Lima. Robot task plan representation by petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, 33(4):337–360, 2012.
6. Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg. *Lectures on concurrency and Petri nets: advances in Petri nets*, volume 3098. Springer, 2004.
7. Alessandro Farinelli, Nicoló Marchi, Masoume M. Raeissi, Nathan Brooks, and Paul Scerri. A mechanism for smoothly handling human interrupts in team oriented plans. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 377–385, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.
8. Francesco Maria Delle Fave, Alex Rogers, Zhe Xu, Salah Sukkarieh, and Nick Jennings. Deploying the max-sum algorithm for coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 469–476, May 2012.
9. M. Ani Hsieh, Anthony Cowley, James F. Keller, Luiz Chaimowicz, Ben Grocholsky, Vijay Kumar, Camillo J. Taylor, Yoichiro Endo, Ronald C. Arkin, Boyoon Jung, Denis F. Wolf, Gaurav S. Sukhatme, and Douglas C. MacKenzie. Adaptive teams of autonomous aerial and ground robots for situational awareness. *Journal of Field Robotics*, 24(11-12):991–1014, 2007.
10. Kurt Jensen and Lars M. Kristensen. *Coloured Petri nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.
11. Gal A. Kaminka and Inna Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 108–113, 2005.
12. Jamie King, R. K. Pretty, and Ray G. Gosine. Coordinated execution of tasks in a multiagent environment. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(5):615–619, 2003.

13. Limor Marciano. CPNP: Colored petri net representation of single-robot and multi-robot plans. Master's thesis, Bar Ilan University, 2013.
14. Borhen Marzougui, Khaled Hassine, and Kamel Barkaoui. A new formalism for modeling a multi agent systems: Agent petri nets. *JSEA*, 3(12):1118–1124, 2010.
15. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
16. Illah R. Nourbakhsh, Katia Sycara, Mary Koes, Mark Yong, Michael Lewis, and Steve Burion. Human-robot teaming for search and rescue. *IEEE Pervasive Computing*, 4(1):72–78, 2005.
17. James Lyle Peterson. Petri net theory and the modeling of systems. *PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, NJ 07632, 1981, 290*, 1981.
18. Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. Cpn tools for editing, simulating, and analysing coloured petri nets. In *Applications and Theory of Petri Nets 2003*, pages 450–462. Springer, 2003.
19. Paul Scerri, Balajee Kannan, Pras Velagapudi, Kate Macarthur, Peter Stone, Matt Taylor, John Dolan, Alessandro Farinelli, Archie Chapman, Bernadine Dias, et al. Flood disaster mitigation: A real-world challenge problem for multi-agent unmanned surface vehicles. In *Advanced Agent Technology*, pages 252–269. Springer, 2012.
20. Paul Scerri, D V. Pynadath, and Milind Tambe. Towards adjustable autonomy for the real-world. *Journal of AI Research (JAIR), 2002, Volume 17, Pages 171-228*, 2002.
21. R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1931–1937, Oct 1998.
22. K Suzanne Barber, Anuj Goel, and Cheryl E Martin. Dynamic adaptive autonomy in multi-agent systems. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(2):129–147, 2000.
23. Katia Sycara, Joseph Andrew Giampapa, Brent K Langley, and Massimo Paolucci. The retsina mas, a case study. In Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omici, and Jaelson Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, volume LNCS 2603, pages 232–250. Springer-Verlag, Berlin Heidelberg, July 2003.
24. Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, pages 83–124, 1997.
25. C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. In F. Schneider L. Parker and A. Schultz (editor), editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 3. Springer, 2005.
26. Prajna Devi Upadhyay, Sudipta Acharya, and Animesh Dutta. Task petri nets for agent based computing. *INFOCOMP Journal of Computer Science*, 12(1):24–35, 2013.
27. Abhinav Valada, Prasanna Velagapudi, Balajee Kannan, Christopher Tomaszewski, George Kantor, and Paul Scerri. Development of a low cost multi-robot autonomous marine surface platform. In *Field and Service Robotics*, pages 643–658. Springer, 2014.
28. Manuela M. Veloso, Martha E. Pollack, and Michael T. Cox. Rationale-based monitoring for continuous planning in dynamic environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 171–179, Pittsburgh, PA, June 1998.
29. F. Y Wang, K.J. Kyriakopoulos, A. Tsolkas, and G.N. Saridis. A petri-net coordination model for an intelligent mobile robot. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(4):777–789, Jul 1991.
30. Jijun Wang and M. Lewis. Human control for cooperating robot teams. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pages 9–16, March 2007.
31. Dianxiang Xu, Richard Volz, Thomas Ioerger, and John Yen. Modeling and verifying multi-agent behaviors using predicate/transition nets. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 193–200. ACM, 2002.
32. V.A. Ziparo, L. Iocchi, PedroU. Lima, D. Nardi, and P.F. Palamara. Petri net plans. *Autonomous Agents and Multi-Agent Systems*, 23(3):344–383, 2011.