A Mechanism for Smoothly Handling Human Interrupts in Team Oriented Plans

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

Nathan Brooks, Paul Scerri
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA USA
nbb@cs.cmu.edu
pscerri@cs.cmu.edu

ABSTRACT

Team oriented plans have become a popular tool for operators to control teams of autonomous agents (or 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 interrupt, the operator would usually need to restart the team plan to ensure its success. In this paper, we present an approach to encoding how unexpected interrupts can be smoothly handled within a team plan. Building on a team plan formalism that uses Colored Petri Net, 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, our interrupt mechanism decreases the time to complete the mission (up to 48% reduction) and decreases the operator load (up to 80% reduction in number of user actions).

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; I.2.9 [**Robotics**]: Operator Interfaces

General Terms

Experimentation

Keywords

Plan Monitoring; Petri Net; Cooperative Robotic Watercraft

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 to environmental monitoring to military operations, approaches are emerging to allow small numbers of humans to control teams of robots for achieving complex objectives. A common way of doing this is via team plans [1, 2, 3], which allow an operator to interact via high level objectives and plans and allow 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 sensitively 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 in bodies of water [4]. In such applications, one or a small number of experienced operators, perhaps water scientists, are managing between five and 25 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.

We specifically adopt an approach for creating team plans with Petri Nets to allow specification of complex, parallel and multi-stage plans. To deal with external interruptions, two special events are created, indicating the start of two categories of interruptions, but not specifying 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 in the plan. Depending on the nature and timing of the interaction, relative to the specific context of the plan, the expressive approach allows 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

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

how to handle interruptions at a particular place in the plan and encode efficient and effective resumptions.

In some interesting plans, there are specific roles for specific assets. Our team plan approach captures this by using a Colored Petri Net formalism and allowing the different colored tokens to represent specific roles. For such plans, we have a special event type which captures which role is being interrupted when the interruption occurs. The plan can then be designed to react differently to different roles being interrupted. For example, interruptions of non-critical roles might be ignored completely, while when one of a set of tightly coupled roles is interrupted, the other roles might be put into some sort of holding pattern. The key is that the design approach provides the representational power to handle the 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 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 mission and up to a 80% reduction in operator load.

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 (states of the system) or transitions (system events), 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 markings of a PN evolves based on the firing behavior of the transitions. A transition t can fire whenever it is enabled (i.e., 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 Net (CPN) [5] extends 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 depends 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 has been addressed from different perspective throughout the years [7, 8, 9]. In particular here we focus on approaches that are most relevant to our work as they are explicitly designed for multi-agent systems. Hence, we discuss Belief Desire Intension (BDI) architectures [1, 2] and approaches based on Petri Net [10, 3].

Two successful BDI-based frameworks are STEAM and BITE which enable a coherent teamwork structure for multiple agents. The key aspect of STEAM [1] are team operators which are based on the Joint Intentions Theory, introduced by [11]. 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 [2], specifies a library of social behaviors and offers different synchronization protocols that can be used interchangeably and mixed as needed. However, while both this 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.

One of the first approaches based on Petri Net for multi-agent plan monitoring was proposed by King et al. [10]. 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) [3] . 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

[1]See for example CPN *Tools* [6]

proposed in [3] 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.

3. THE COOPERATIVE ROBOTIC WATERCRAFT FRAMEWORK

This work focuses on a system of robotic boats developed as part of the Cooperative Robotic Watercraft project [4]. In this Section we describe the team plan specification language used in such system (SAMI) and provide a brief overview of the whole system.

3.1 SAMI Petri Net

The main components of the SAMI Petri Net specification language are illustrated in a sample plan in Figure 1. The language is based on Colored Petri Nets and Hierarchical Petri Nets and has several 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 directives. In particular, the plan shown in Figure 1 reports places and transitions with associated output (OE) and input events (IE). When the plan starts the output event *Operator Select Robot List* is triggered asking the operator to select the list of boats that will participate in the plan. When the operator performs this action the input event *Operator Selects Boat List* is triggered, the corresponding transition will then fire moving the *relevant* tokens (i.e., the tokens that correspond to the selected boats) to the next place. The plan progresses in a similar way until the tokens reach the last place (i.e., all selected boats completed their path). When this happens the mission is over and the plan is no longer active.

Events fall under two categories: *output events* and *input events*. Output events are added to places in the Petri Net and contain commands or requests that are sent to human operators, robots, or agents. When a token(s) enter a place, all the output events on the place are triggered. The registered handler for that class of output event is sent the output event along with the tokens that just entered the place. For output event classes that contain data fields, there are 4 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 when the plan is instantiated (the location boats should move to have their battery swapped). (3) Value defined by operator at run-time when the plan reaches place containing event (a safe temporary position for robots to move to in order to avoid an incoming manned boat). (4) Variable name defined by developer at plan design time, whose value is retrieved at run-time when plan reaches the place containing the event (a variable to retrieve the path planning results from a planning agent).
Input events are put on transitions and contain information received from human operators, robot proxies[2], or agent services, which perform assistive functions such as path planning, task allocation, and image processing. Input events on a transition are typically responses to an output event on a place preceding the transition. For input events 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 stored and accessed by output events. When an input event is received by the plan execution engine, it copies each piece of information held by the input event to the corresponding variable name that was specified by the plan developer. Input events also contain "relevant proxy" and "relevant task" fields, which contain the identities of the proxy(s) or task(s) that sent or triggered the input event.

Markup Each plan is "marked up" with context clues attached to individual events which provide information to the run-time GUI about which UI 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. Each markup has a number of options and variables that the petri net developer must specify. UI 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 UI 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.

Tokens The Token class has 4 fields: a name (String), a token type (TokenType), a proxy (ProxyInt), and a task (Task). There are 3 TokenTypes: *Generic* tokens have no defined proxy or 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. Representing proxies and tasks using tokens allows for multi-robot plans with arbitrary numbers of team members to be constructed and visualized compactly, compared to having individual Petri Net for each member of the team.

Edge Requirements In a standard Petri Net, incoming edges specify the tokens required for a transition to fire, which are then consumed, and outgoing edges specify the tokens that are added to the connected place. SAMI Petri Net edge labels have additional options designed to provide additional power. Labels on incoming edges specify tokens that must be present in the connected place in order for the connected transition to fire, but does not remove them upon firing as it could cause undesired interruption of behavior controlled by output events in the connected place. Labels on outgoing edges specify tokens that should be taken or consumed from the incoming places preceding the connected transition and/or tokens that should be added to the connected place. Taking a token removes it from its old place and adds it to the place connected to the outgoing edge. Consuming a token simply removes it from its old place, while adding a token will not modify the old place and just add a copy of the token to the connected place. In addition to generic tokens and 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 spec-

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

ifies 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 and another task with a sequential completion constraint can begin.

Sub-missions The SAMI team plan language supports hierarchical team plans, allowing a "sub-mission" to be added on a place. When tokens enter the place an instance of the sub-mission is started, placing the tokens that entered the place into the start place of the sub-mission. Until the sub-mission finishes executing, transitions in the parent mission leaving the place holding the sub-mission are prevented from firing. Sub-missions can return values, return tokens, and write to variables shared with its parent mission, allowing sub-missions to reduce repeated creation of common Petri Net sequences and increase readability of the petri net.

3.2 System design

Figure 2 shows an airboat style robot, although differential drive propeller versions have also been developed. 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 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. 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 along 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 SAMI Petri Net 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 markups they support, allowing a custom "interaction panel" to be constructed for each event requiring operator input.

In order to assist the petri net developer, an intelligent plan editing tool named DREAAM was created. The editor was designed with three potential limitations of the plan language in mind: overwhelming visual clutter and developer errors resulting in invalid Petri Net or unexpected run-time behavior. DREAAM contains different visualization modes which selectively hide and compress sections of nets based on different tasks the developer may be performing. "Assistant agents" check for violations of Petri Net rules and flag errors, such as incomplete graphs and unlabeled start/end places, and warnings, such as suspicious edge labels.

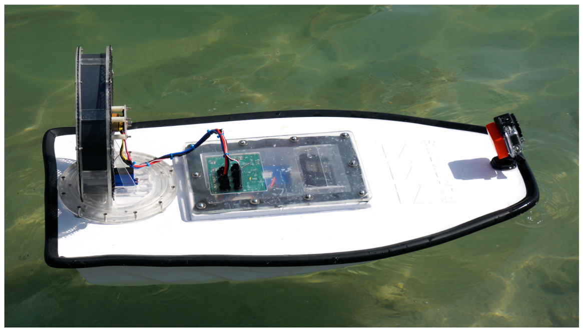


Figure 2: Airboat robot platform

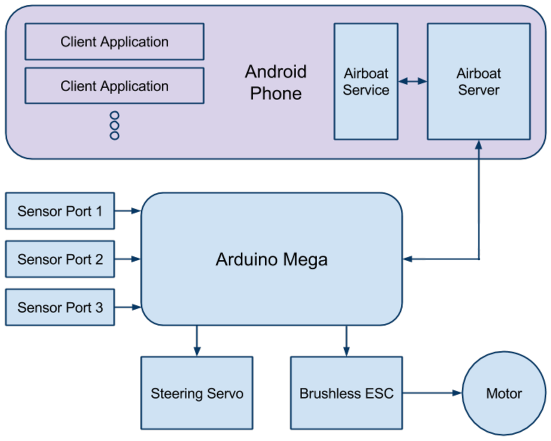


Figure 3: CRW system architecture

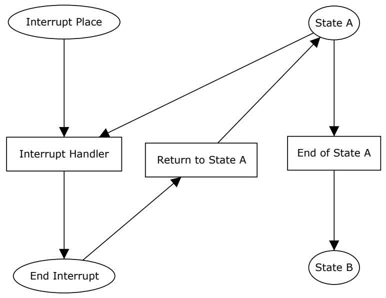


Figure 4: Interrupt implementation with Petri Net.

4. THE INTERRUPT MECHANISM

In this Section, we describe the basic ideas of the interrupt mechanism for Petri Net 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 Modelling 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 [12].

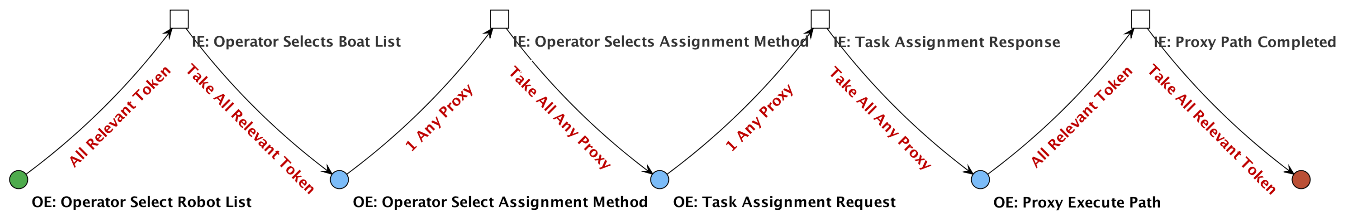

IE: Operator Selects Boat List
IE: Operator Selects Assignment Method
IE: Task Assignment Response
IE: Proxy Path Completed
All Relevant Token
Take All Relevant Token
1 Any Proxy
Take All Any Proxy
1 Any Proxy
Take All Any Proxy
All Relevant Token
Take All Relevant Token
OE: Operator Select Robot List
OE: Operator Select Assignment Method
OE: Task Assignment Request
OE: Proxy Execute Path

Figure 1: SAMI Petri Net "Cooperative Location Visit" plan (without the interrupts)

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 (*StartInterruptTransition*) and, iii) a transition that determines the end of the interrupt procedure (*EndInterruptTransition*). Now, consider a general plan that we represent with a *Source Place*, indicating the state of the system that could receive an interrupt, a general transition, indicating a general part of a plan, and a *Destination Place*, indicating the state of the system that should be reached when the interrupt handling procedures terminates (notice that the source and destination places could be the same). Figure 5 shows the structure we propose to add an interrupt to such general plan. As the figure shows, the *StartInterruptTransition* and the *EndInterruptTransition* are connected by a *Submission Interrupt Place* which represents a general sub-plan that models the appropriate interrupt handling behaviour. After the execution of the submission all the tokens that comes out from the submission move to the destination place of the interrupt, and restore the normal behaviour of the plan.

The structure described above can represent two types of interrupts: a *proxy* interrupt (see Figure 5(a)) and a *general* interrupt (see Figure 5(b)). As shown in Figures 5(a) and 5(b), the structure to realize these two types of interrupts is the same, however, the events attached to places/transitions and the labels to the edges of the net are different. In what follows we describe these two interrupt types in more detail.

The 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 *ProxyInterruptEvent*, which is an output event[3]. The *ProxyInterruptEvent* encapsulates the information regarding which proxies should be involved in the event. Such information is used by the *Start Interrupt Transition* to consume only the relevant tokens from the *Source Place* and transfer them to the *Submission Interrupt Place*. Consequently, only the tokens specified by the *ProxyInterruptEvent* will stop their current mission to execute the interrupt submission. Such relevant tokens are selected with a mission specific procedure, and this often requires a user interaction (i.e., the user directly selects which platforms should execute the interrupt submission).

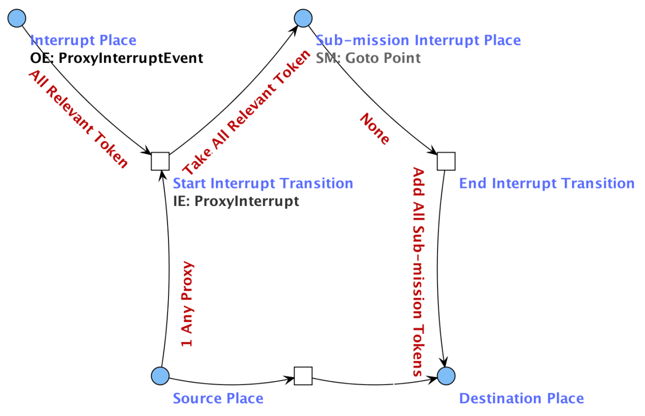
General Interrupt The *general* interrupt is a team-level interrupt that is not specific to a particular platform. The *general* interrupt represents a situation where all robotic-boats should perform a particular procedure, e.g., stop all current plans and go to a safe position as a manned boat is approaching.

In contrast to the *proxy* interrupt, the *general* interrupt will remove all tokens present in the *Source Place* and transfer them to the submission. Hence, the event generated by the *interrupt place* is a different output event, named *InterruptEvent*. 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 *StartInterruptTransition* requires a *generic* token (and not a proxy token) and it will transfer all the *proxy* tokens from the *Source Place* to the *Submisison Interrupt Place*. 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.

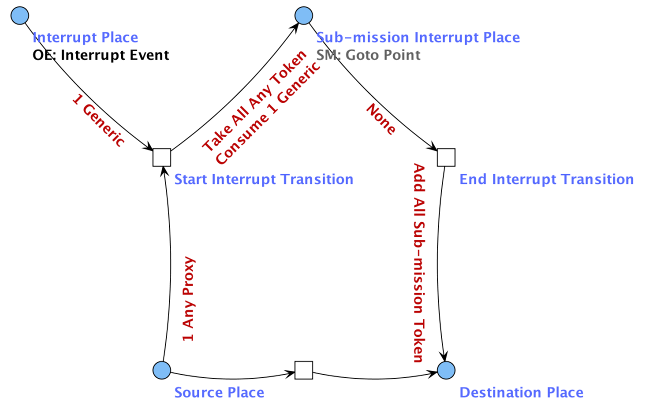
4.3 Using interrupt in multi-agent plans

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 so 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 [13]. The method assign 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

[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



(a) Proxy Interrupt



(b) General Interrupt

Figure 5: Types of interrupt implemented in the SAMI framework.

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 plan, the general interrupt handles a situation where the user decides to temporarily stop the current mission 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 *ProxyExecutePath* place and goes back to the same place. When the interrupt triggers, all the tokens present in the *ProxyExecutePath* place are transferred to the submission place. The submission associated to the interrupt (not shown in this picture) sends all the boats to a specific safe assemble position and then waits for an operator input to re-start the previous mission. When the operator decides that the dangerous situation is over, the boats are sent back to the *ProxyExecutePath* place where they resume executing their mission, maintaining their previous location assignment.

In contrast, the proxy interrupt allows the operator to stop the execution of a selected subset of the boats without interfering with the mission 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 critical level of the battery. The proxy interrupt moves the set of selected proxies to the submission place while the others will continue their execution. In our exemplar plan, the submission associated to the interrupt is a recharge submission that stops the current plans of the selected proxies and sends the relevant proxies to a recharge station, where the batteries are replaced. Then the proxies re-start their previous mission.

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

5. EMPIRICAL RESULTS

In this section we present a quantitative evaluation of our approach to team plan monitoring for the CRW framework. 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 mechanism, in terms of task specific performance as well as operator load, with respect to aborting the plan whenever an external event takes place.

As a first step, we consider the CLV plan discussed in Section 4.3 considering the "interrupt" version that encodes the interrupt within the plan (reported in Figure 6) and the "standard" version without the interrupts (reported in Figure 1). Next, we define two possible external events: i) *general alarm* and ii) *temporary boat pull-out* that should be handled at execution time by a human operator. We then simulate the execution of both versions of the CLV plan with both events, measuring indicators of task specific performance and operator work load. When we execute the standard plan and one of the external events takes place, the human operator must abort the entire mission execution, execute the plan that can handle the specific event, and then restart the original plan.

External Events As for the external events, the *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 (e.g., executing 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 events while a CLV plan is running. In particular, we fix the number of events to happen and distribute them randomly during the mission execution,

In contrast, the *temporary boat pull out* represents an event 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

[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 locations as the one that is closest to the current location. At the beginning the current location is the boat position.

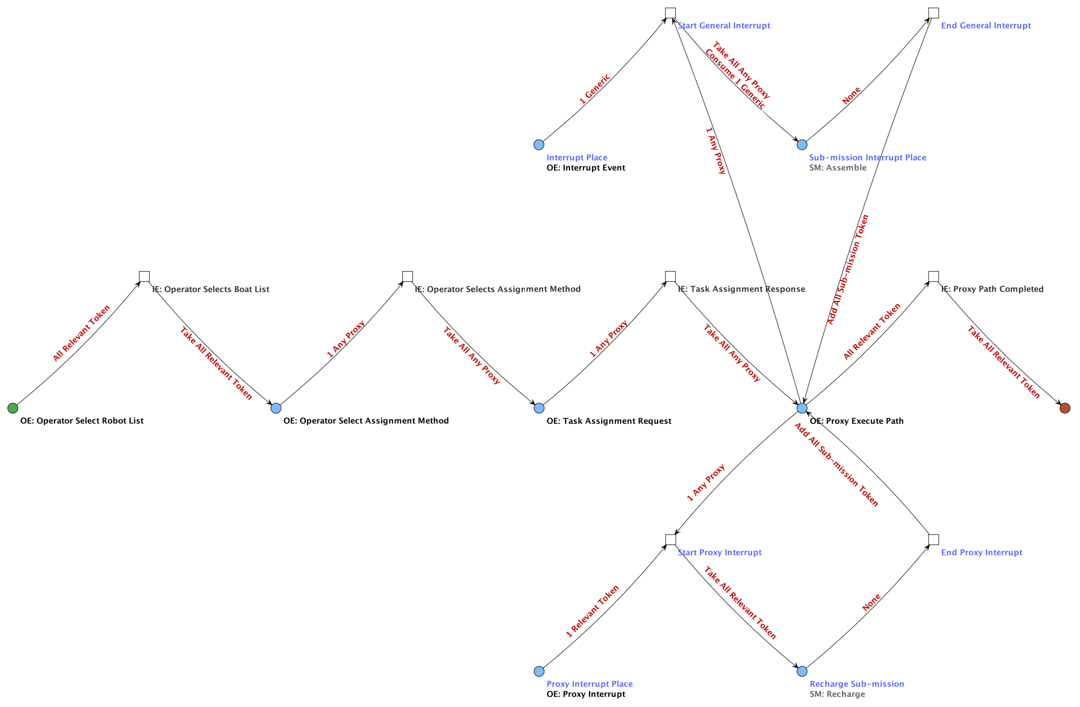


Figure 6: The Cooperative Location Visit plan specified in the SAMI framework, with both interrupts (general and proxy).

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.

When we execute the plan with the interrupt mechanism in place, we assume that whenever an external event happens, the operator will trigger the corresponding interrupt. For example, when we execute the CLV 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 the external events. 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 plan without the interrupt, we assume that the human operator will abort the current plan, start the plan for the specific interrupts and, finally, start a new plan to complete the original mission once the interrupt has been handled. Note that, when we start a new plan the operator must re-insert all information required by the plan, such as the locations to be visited. 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 (this reduces the number of interactions in favor of the standard approach). Moreover, we assume that the operator will start new plans only after the recharge mission has been completed. This means that the other boats will remain idle for the recharge mission duration. Another execution model could start new plans for the other boats while the recharging boat accomplishes its mission and then restart the plan with all the boats once the recharge process is over. While this would reduce the overall time of the mission it would significantly increase the number of interactions with the user.

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

5.2 Results

Table 1 reports results obtained for the CLV plan and the boat pull out event. 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 (10,20) 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 con-

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.

figurations that show a statistically significant difference between the values of the means[5].

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 mission as well as for the operator workload. In more detail, focusing on the time to complete the mission, 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 mission 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 battery discharge process depends on the travel distance, and since the interrupt model minimizes the idle time of the boats, each boat will on average travel more (this is confirmed by the positive gain in the total time).
Table 2 reports results achieved for the CLV plan and the general alarm event. We considered the same number of boats and number of tasks, and we vary the number of alarm events that will appear during the mission (1,3). As before, we report the average values of the gain and the standard error of the mean.

For what concerns 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 mission 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 mission can be safely re-started. Hence, the actions that the boats perform when aborting a plan are very similar to the interrupt handling procedure. Notice that, 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.

Finally, a video showing an exemplar execution of the CLV plan presented in Figure 6 is reported here[6]. The video shows that, when the general interrupt is triggered all the boats move through the interrupt branch, and enters a recovery sub-mission that sends them all to a safe assemble location. When the alarm is over the boats restart their previous missions. 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 mission, it returns to finish executing its previous mission.

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.

6. CONCLUSIONS

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 interrupt 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 build on the SAMI framework, which proposes the use of Colored Petri Net for specifying team plans, and we define 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.

Acknowledgments

This work is funded by the Qatar National Research Fund NPRP grant 4-1330-1-213.

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

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

REFERENCES

[1] M. Tambe, "Towards flexible teamwork," *Journal of Artificial Intelligence Research*, pp. 83–124, 1997.

[2] G. A. Kaminka and I. 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*, 2005, pp. 108–113. [Online]. Available: http://www.aaai.org/Library/AAAI/2005/aaai05-018.php

[3] V. Ziparo, L. Iocchi, P. Lima, D. Nardi, and P. Palamara, "Petri net plans," *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 3, pp. 344–383, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10458-010-9146-1

[4] P. Scerri, B. Kannan, P. Velagapudi, K. Macarthur, P. Stone, M. Taylor, J. Dolan, A. Farinelli, A. Chapman, B. Dias *et al.*, "Flood disaster mitigation: A real-world challenge problem for multi-agent unmanned surface vehicles," in *Advanced Agent Technology*. Springer, 2012, pp. 252–269.

[5] K. Jensen, *Coloured Petri nets: A high level language for system design and analysis*. Springer, 1991.

[6] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen, "Cpn tools for editing, simulating, and analysing coloured petri nets," in *Applications and Theory of Petri Nets 2003*. Springer, 2003, pp. 450–462.

[7] M. M. Veloso, M. E. Pollack, and M. T. Cox, "Rationale-based monitoring for continuous planning in dynamic environments," in *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, Pittsburgh, PA, June 1998, pp. 171–179.

[8] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, Oct 1998, pp. 1931–1937.

[9] F. Y. Wang, K. Kyriakopoulos, A. Tsolkas, and G. Saridis, "A petri-net coordination model for an intelligent mobile robot," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 4, pp. 777–789, Jul 1991.

[10] J. King, R. K. Pretty, and R. G. Gosine, "Coordinated execution of tasks in a multiagent environment," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 33, no. 5, pp. 615–619, 2003. [Online]. Available: http://dx.doi.org/10.1109/TSMCA.2003.817387

[11] P. R. Cohen and H. J. Levesque, "Teamwork," *Special Issue in cognitive Science and Artificial Intelligence*, pp. 487–512, 1991.

[12] J. Desel, W. Reisig, and G. Rozenberg, *Lectures on concurrency and Petri nets: advances in Petri nets*. Springer, 2004, vol. 3098.

[13] C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig, "The generation of bidding rules for auction-based robot coordination," in *Multi-Robot Systems: From Swarms to Intelligent Automata*, F. S. L. Parker and A. S. (editor), Eds., vol. 3, no. 14. Springer, 2005.