Situational Awareness and Mixed Initiative Markup for Human-Robot Team Plans

Nathan Brooks

CMU-RI-TR-17-64

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Paul Scerri, *Chair*
Illah Nourbakhsh
Reid Simmons
Julie Adams, *Oregon State University*

August 2017

Copyright © 2017 Nathan Brooks

Abstract

As robots become more reliable and user interfaces (UI) become more powerful, human-robot teams are being applied to more real world problems. Human-robot teams offer redundancy and heterogeneous capabilities desirable in scientific investigation, surveillance, disaster response, and search and rescue operations. Large teams are overwhelming for a human operator, so systems employ high level team plans to describe the operator's supervisory roles and the team's tasks and goals. In addition, UIs apply situational awareness (SA) techniques and mixed initiative (MI) invocation of services to manage the operator's workload. However, current systems use static SA and MI settings which cannot capture changes in the plan's context or the overall system configuration. The configuration for one domain, device, environment, or section of a plan may not be appropriate for others, limiting performance.

This thesis addresses these issues by developing a team plan language for human-robot teams and augments it with a situational awareness and mixed initiative (SAMI) markup language. SAMI markup captures SA techniques for UI components, MI settings for decision making, and constraints for algorithm selection at specific points in a team plan. In addition, we identify properties of the team plan language and use them to develop semantic and syntactic software agents which aid plan development.

To test the team plan language and markup's ability to capture complex behavior and context specific needs, we design several experiments in simulation and deploy a large team of autonomous watercraft. Run-time statistics and the team's ability to adapt to challenges "in the wild" are used to evaluate the effectiveness of the marked up language.

To assess the learnability of the language by non-experts, a user study evaluating a series of self-guided lessons is designed. Users with exposure to computer science concepts complete training material during which task performance and interviews are used to assess the effectiveness and scalability of the material.

These contributions demonstrate an approach to improve the accessibility of human-robot teams and their performance in complex environments.

Acknowledgments

In a dissertation about team plans, it goes without saying that success is achieved through the combined efforts of many agents. First I would like to thank my advisor, Paul Scerri, for the countless hours spent mentoring, critiquing papers, debugging code, and deploying robots. I would also like to thank my committee members, Illah Nourbakhsh, Reid Simmons, and Julie Adams, for their insight into experimental design. So much of this work would not have been possible without the dedicated efforts, both intellectual and physical, from the rest of the lab: Chris, Jason, Alex, Pras, Abhinav, Laura, and Balajee.

CMU is fortunate to have many dedicated people who work tirelessly to remove obstacles to success both inside and outside of the lab. I have especially benefitted from the support of Suzanne, Jean, Alan, and Sumitra in the RI and Dawn and Tiffany in the SLICE office. I am also extremely grateful for my friends at CMU Qatar who helped me adjust to life in Doha and track down hard-to-find parts.

Much of this thesis is the result of fruitful collaboration with friends and colleagues. I would especially like to thank Alessandro, Nicol, and Masoume at the University of Verona and Elan, Sean, Richard, Ewart, Athena, and Dan at Perceptronics Solutions, Inc.

This document would certainly not exist without my many friends who have inspired and encouraged me. I have been fortunate to live with some incredible people at Robot House and the Aurelian Empire, as well as many others who have graciously offered me housing these last months. I have many fond memories with my friends in the Dinner Train, RoboOrg, and CMU Explorers Club working on assignments, practicing talks, solving puzzles, and exploring the outdoors. As a PhD student, there are boundless opportunities for intellectual growth, making it easy at times to neglect growth in other areas. For this I am truly thankful for the Explorer's Club of Pittsburgh, whose commitment and camaraderie are unmatched. Finally, it is rare for a team plan to have truly predictable behavior, and for that I am especially grateful to my family for their unwavering encouragement and support.

Funding

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

Contents

1 Introduction **1**
1.1 Thesis Contributions . . . 3
1.2 Reader's Guide . . . 5

2 Background **7**
2.1 Markup Languages . . . 7
2.2 Human-Robot Interaction . . . 11
2.2.1 GUI Building . . . 11
2.3 Planning Algorithms . . . 13
2.3.1 Autonomy . . . 14
2.4 Team Plan Representation . . . 15
2.4.1 Belief Desire Intention . . . 15
2.4.2 Finite State Machines . . . 16
2.4.3 Petri Nets . . . 18
2.4.4 Colored Petri Nets . . . 21
2.4.5 Language Design . . . 22

3 Language Syntax **24**
3.1 SPN Definition . . . 24
3.1.1 Summary . . . 30

4 Language Properties **34**
4.1 Language Deviation . . . 34
4.1.1 Events . . . 34

4.1.2 Color Sets 35
4.1.3 Out Edge Requirement Actions 36
4.1.4 In Edge Requirement Quantity 39
4.1.5 Sub-missions 41
4.2 Behavioral Analysis 41
4.2.1 Reachability Graphs 41
4.2.2 Coverability Graphs 42
4.3 Graph Based Assistants 43
4.3.1 Reachability 43
4.3.2 Boundedness 43
4.3.3 Other Assistants 43
4.3.4 Summary 44

5 GUI **45**
5.1 DREAMM IDE 45
5.1.1 Event Wizard 47
5.2 GUI 51
5.2.1 CRW GUI 51
5.2.2 AIMS GUI 53
5.2.3 Phase Chart 55
5.2.4 Component Construction 57
5.3 Meta-markup 63
5.3.1 Summary 67

6 Field Deployment **69**
6.1 Boat Development 69
6.1.1 Winter 2012 Model 71
6.1.2 Winter 2013 Model 72
6.1.3 Winter 2014 Model 74
6.2 Test Sites 77
6.3 Deployed Plans 78
6.3.1 Connect Boat 78
6.3.2 Connect and Station Keep 80

6.3.3 Explore Area 87
6.3.4 Communications Monitoring 93
6.3.5 Communications Measurements 98
6.3.6 Summary 100

7 Interrupt Evaluation **104**
7.0.1 PN Modeling of an Interrupt 104
7.0.2 SPN Modeling of an Interrupt 104
7.0.3 Using Interrupts 106
7.1 Empirical Results 108
7.1.1 Empirical Methodology 110
7.1.2 Quantitative Results in Simulation 116
7.1.3 Validation on Robotic Platforms 120
7.1.4 Summary 122

8 Markup Evaluation **123**
8.1 Design 123
8.2 Results 125
8.2.1 Summary 127

9 User Study **129**
9.1 Study Design 130
9.1.1 Capturing the Demographic 130
9.1.2 Designing the Material 131
9.2 Study Results 143
9.3 Analysis 146
9.3.1 Demographic 146
9.3.2 Language Scalability 146
9.3.3 Language Learning 147
9.3.4 Language Application 148
9.3.5 Summary 150

10 Discussion **152**
10.1 Summary 152
10.2 Future Topics 154
10.2.1 Geographic Markup 154
10.2.2 Designer Development 154
10.2.3 Markup Transferability Across Devices 156
10.2.4 Decentralized Execution 156

Appendices **i**
.1 Source code ii
.2 CRW library ii
.2.1 Output event list ii
.2.2 Input event list iv
.2.3 Markup list vi
.3 Field deployment data viii
.4 User study material ix
.4.1 DREAMM Lesson Modes ix
.4.2 Tutorial Slides x

List of Figures

2.1 A team plan representation (dashed box) with execution context (yellow call-outs) . . . 8
2.2 LaTEX code . . . 9
2.3 HTML code . . . 10
2.4 CSS code . . . 10
2.5 Some "themes" offered by Wordpress . . . 11
2.6 XML data . . . 11
2.7 Netbeans Swing GUI Builder . . . 12
2.9 An explosive ordinance disposal (EOD) MissionLab plan (from [57]). . . . 17
2.10 Key types of PNP structures taken from [201] . . . 19
2.11 The Score_Goal task plan PN (from [40]) . . . 20
2.12 The dynamically allocated Organize Room CPNP (from [128]). Some repeated arc expressions are omitted for compactness. . . . 22

3.1 SAMI Petri Net "Cooperative Location Visit" plan (without the interrupts). The starting place is colored green and the end place is colored red . . . 29
3.2 SAMI Petri Net showing a partial execution of the "CLV" plan (without the interrupts) . . . 31

4.1 PNP actions taken from [201] . . . 34
4.2 Moving SPN events to only use transitions . . . 35
4.3 Color declarations for SPN token types . . . 36
4.4 Conversion of "Consume" out edges in an SPN . . . 37
4.5 Conversion of "Consume" out edges in an SPN . . . 38
4.6 Conversion of "Add" out edges in an SPN . . . 38

4.7 Conversion of an SPN with events and edge requirements 38
4.8 Conversion of "Take" out edges in an SPN 39

5.1 DREAMM SPN editor 46
5.2 Using the recovery visualization mode in DREAMM 49
5.3 Modifying variable for an output event 50
5.4 SAMI system architecture design 52
5.5 CRW GUI 54
5.6 AIMS GUI 56
5.7 Phase markup for UAV search plan 58
5.8 "Get locations" SPN section and phase chart 59
5.9 "Task assignment" SPN section and phase chart 60
5.10 "Task execution" SPN section and phase chart 61

6.1 Boat propulsion methods 70
6.2 Boat propulsion methods 72
6.3 2014 boat components 76
6.4 2014 boat components 76
6.5 Deployment locations 77
6.6 Deployment setup 78
6.7 Plan invocations during a 180 minute deployment at Katara Beach 79
6.8 Connect Boat SPN 80
6.9 Alphabetized boat ID list interpreted during Connect Boat SPN 81
6.10 Connect and Station Keep SPN 84
6.11 Video frames from Connect and Station Keep video 86
6.12 Boat paths during connect and station keeping 87
6.13 Explore Area SPN 90
6.14 Speedboat SPN used as sub-mission in Explore Area 91
6.15 Explore Area SPN executing 92
6.16 Boat paths during lawnmower exploration 93
6.17 Communications Monitoring SPN 96
6.18 Identifying WiFi network failure 97
6.19 Boat paths during router failure 98

6.20 Distance (in meters) at which boats can maintain a stable connection to the base station antenna (yellow marker) while floating in the water (top semicircle) versus sitting on the shore (bottom semicircle) 99
6.21 Non-expert sketch of a desired SPN 101
6.22 Two waypoint initial implementation of sketch 102

7.1 Interrupt implementation with Petri Net. 105
7.2 Types of interrupt implemented in the SPN framework. 107
7.3 The Cooperative Location Visit plan specified in the SPN framework, with both general and proxy interrupts. 109
7.4 CLV plan with the interrupt for traverse dangerous area 114
7.5 A picture of the connect and station keep experiment. The image shows a subset of the platforms and the current state of the SPN representing the connect and station keep plan. The interrupt portion of the plan is visible in the top part of the picture and the firing transition (highlighted in purple) is the one that starts the interrupt to change the position where boats should perform station keeping. 121

8.1 Avoiding a hazard while mapping an area 124
8.2 SPNs used during markup evaluation simulation 126

9.1 Recruitment Survey 132
9.2 Incremental complexity in Connect and Station Keep SPN 133
9.3 Some slides from Lesson 6: Robot tokens 136
9.4 Station keeping SPN 137
9.5 Measurement collection SPN 138
9.6 Asynchronous panorama SPN 140
9.7 Synchronous panorama SPN 141
9.8 Retroactive probing dialogue 142
9.9 Confusion about looping 149

10.1 Sample tree for Designer workflow 155

List of Tables

1.1 Contributions discussed in each thesis chapter 6

6.1 Runtime statistics for Connect Boat over 2 days 80
6.2 Runtime statistics for Connect and Station Keep SPN over 2 days 83
6.3 Runtime statistics for Explore Area SPN over 2 days 89

7.1 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.) 116
7.2 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. 117
7.3 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). 117
7.4 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. 118

8.1 Percent gain for statistics comparing markup (M) version of plan to generic (G) version of plan. (-) indicates a negative value is desired, (+) indicates a positive number is desired. 127

9.1 Completion time for each lesson and each participant (hour: minutes: seconds) Bolded, blue durations indicate above average; red durations indicate below average 144
9.2 Number of days elapsed since previous lesson: Bolded, red quantities are greater than 2 days 145
9.3 Number of quizzes answered correctly for each lesson and each participant: Bolded, red numbers indicate at least one answer was incorrect. 145
9.4 Number of jobs performed correctly for each lesson and each participant: Red numbers indicate at least one job was not performed correctly 146

1 Summary of CRW output events iv
2 Summary of CRW input events vi
3 Summary of CRW markups viii
4 DREAMM lesson modes x

List of Algorithms

1 Checks if a transition should be enabled 30
2 Fires an enabled transition 32
3 Handles tokens entering a place 33
4 Produces a component meeting certain requirements 64
5 Direct score 65
6 Recursive score 66

Chapter 1

Introduction

Multi-robot teams offer powerful capabilities for many real world problems, including disaster response [140, 113, 26, 157, 140, 172, 114, 112], environmental monitoring [187, 123, 37, 52, 47], exploration [195], agriculture management [12, 12], network formation [38, 85, 142, 98, 18], and search and rescue [73, 76, 75, 155, 14]. Robots can achieve high level goals that are not possible for a single robot by coordinating the execution of interdependent *tasks*. Heterogeneous teams [86] allow for a wide range of tasks to be performed and enables performance gains by selecting the most capable robot for particular tasks. To describe the relationships between tasks and coordinate their execution, an *expert* describes this information offline in a *team plan* [185, 164, 179]. When a team plan is run, it assigns tasks to robots and the robots in turn invoke code to achieve each task. In the event of a robot failure, team plans can describe how to reassign a robot's tasks to functional teammates. This reassignment allows larger teams to be more robust to failures compared to a smaller team.

While multi-robot teams allow for a high level of performance and robustness, they require complex software architecture [131, 56, 109]. At a low level is the code which runs on individual robots, which may feature the latest and greatest algorithms for path planning, localization, mapping, and task allocation for a particular scenario and set of assumptions. At a high level is the team plan, which is commonly written in a specialized team planning language capable of describing tasks and their relationships in a compact format. Team planning languages often provide tools for finding logical errors which would result in a deadlock of the team or undesired behavior. To bridge these two layers, *interpreter* code is needed to translate an assigned task in a team plan into code invocation on a robot. These architectures typically also require several configuration files to tune parameters, describe the team, and connect the low level code, interpreter code, and team plan tasks.

While many multi-robot teams execute team plans fully autonomously [132], including humans in the team provides many benefits. Human-robot teams can provide better performance and robustness using the human's domain knowledge to handle conditions which

elude sensors, behavior that is hard to recognize, and actions that are difficult to formalize. These *operators* - ecologists, rescue workers, and soldiers - increase the team's flexibility and robustness by providing high level guidance to the team and low-level assistance to robots [2, 106, 99, 5]. Human-robot team plans capture the operator's responsibilities in addition to those of the robots, and the interpreter translates those responsibilities into code invocation on the user interface (UI). However, operators are limited in how many robots they can supervise due to natural limitations in memory, reaction time, and other cognitive resources.

To make the most of the operator's limited resources, application specific UIs employ various techniques and tools to reduce the cognitive load placed on the operator [177]. Such UIs have allowed non-expert operators to successfully control large teams of robots in a number of domains [36, 141, 146, 165, 89, 177, 64]. One method to improve the operator's performance is to improve their internal model of the state of the world and how events will affect it, commonly referred to as situational awareness (SA) [112]. By intelligently controlling if, when, and how information is presented to the operator in the UI, their SA can be improved, reducing the time it takes to process information and make decisions. Other research looks into the best method to convert the user's SA and knowledge into actions by improving components in the UI. By improving the layout and interaction style of components, operators can more efficiently take action. Yet another direction of research focuses on improving the autonomy of individual robots to avoid situations which will require the operator's attention. Use of autonomy is highly situational: autonomy lightens the operator's workload, but reduces their situational awareness and can have negative consequences if the autonomy is unreliable. *Levels of Autonomy* [173, 149] describe a spectrum of collaboration between operator and autonomy: the highest level uses full automation, fully ignoring the operator, and the lowest level assigns full responsibility to the operator, providing no automated assistance. Levels in the middle use *mixed initiative*, where autonomy can make decisions for the operator, but the operator can intervene and adjust the decision themselves at any time. This allows for the operator to be utilized when they are not busy, but does not deadlock the team by requiring their input when they are occupied. Typically, a high level of autonomy is used for low-level decisions with reliable autonomy. For decisions which are high risk, high impact, or have unreliable autonomy, a low level of autonomy may be used. Applying these techniques, human-robot systems often use a human operator at a supervisory level to decide which team plans to run and provide assistance in their execution as needed [31, 32, 68, 69, 135, 136, 65, 148].

In general, design decisions regarding situational awareness, UI layout, and level of autonomy are optimized for a motivating scenario when developing the system, which limits their adaptivity to unexpected events and new applications. As human-robot teams become common tools capable of addressing a variety of scenarios and facing real-world failures, a single set of rules cannot capture the needs of every situation. For example, when deciding the best use of an operator's resources, abnormal behavior for a robotic boat should be

treated differently if it is in a pond versus near the edge of a dam. For the operator GUI, the most appropriate types of information to display and component interaction style also varies by situation. Humphrey [88] compares task performance between two UI map components and finds that a top-down exocentric compass visualization to be better for metric judgment tasks and an in-world egocentric visualization to be better for navigational tasks. For task allocation algorithms, computation time balances against the quality of the solution, which includes how many tasks are successfully allocated and how capable robots are for their assigned tasks [108]. In tasks involving robot movement, allocation algorithms also trade off how quickly the tasks will be completed against how efficiently they are performed. For example, having a single robot perform many tasks in a local region may be efficient, but moving other robots from another region to assist will result in the tasks being completed faster at the expense of additional energy consumption. All of these design decisions require knowledge of the *context* [115, 23, 10, 174, 63, 66] of the operation.

While the team plan is developed knowing exactly how each action contributes to the overall goal and the status of the plan when a specific action is being executed, this context is not captured in the team plan and thus cannot be transferred to the interpreter. To the interpreter, two separate actions of moving from A to B are identical, even if the plan developer added one of the actions knowing it was in the context of an emergency. The key insight of this thesis is the concept of "marking up" a team plan with explicit contextual information which is passed to the interpreter, allowing the system's behavior to be adapted appropriately.

This thesis asserts that a domain independent planning language and markup language capturing context specific human-robot team behavior can be designed which non-experts can be trained to use, which will improve practical deployment.

1.1 Thesis Contributions

This thesis makes 4 contributions:

1: SPN, an team planning extension to CPNs

In our first contribution, we address how to capture domain independent human-robot team plans in a way that is accessible to non-experts. To operate in real-world domains, a complex team planning language is needed to describe coordination of interdependent actions, reassignment of failed robot's tasks, and contingencies for unexpected circumstances. As human-robot teams become common tools in these environments, users may not have an expert available to create or adjust plans, so a language non-experts can learn in a reasonable amount of time is advantageous. In general, an increase in language complexity requires an increase in training to use it, so a careful balance is needed between complexity and learnability. We develop a planning language focusing on the ability to capture human-robot

coordination in a compact, visual format. The language is based on Colored Petri Nets, with several modifications to add representational power in a concise manner. By building off of this framework, the language inherits behavioral verification properties as well as a single representation for both execution and monitoring of designed plans.

2: SAMI, a markup language for team plans
Our second contribution addresses how to add context clues to team plans in a domain flexible manner. One risk when developing the format for team plan markup is overfitting for a particular type of action or domain, building it around concepts which do not translate to other actions or domains. Overly specific markup will result in effectively having a unique set of markup for every action in every domain. However, markup that is too generic may be unable to capture contextual information with sufficient detail to improve performance. We develop a markup language that can be applied to team plans to capture contextual information at specific points. The markup can affect settings in the UI, optimization criteria for algorithm selection, and the level of autonomy used for decision making. Meta-markup is also developed which places limitations on how frequently or to what degree markup can modify the UI.

3: Field observations and lessons learned
Our third contribution is a presentation of plan evolution and lessons learned over several years of field deployment and robot design. To test the ability of the designed team plan and markup languages to capture complex behavior, we write several team plans for a fleet of up to two dozen small, autonomous boats. We contribute lessons learned from deploying large robot teams outside of a laboratory environment. In particular, we discuss contingencies which should be considered when designing team plans for use in the field. Furthermore, we present lessons learned over several iterations of designing and manufacturing the boat platform.

4: Model for design and evaluation of training material
Our fourth contribution addresses how to design and evaluate training material for designing team plans. We create a set of in-depth lessons to teach non-experts how to use the developed language and markup. We present a user study based approach for evaluating the effectiveness of the lessons in achieving several goals. First we consider how to recruit participants representative of the target demographic of the lessons and how to determine the bounds of user backgrounds for which the lessons are effective. After conducting the user study, we discuss measures which capture how well the lessons can be scaled to mass consumption. We also discuss methods and metrics for evaluating how well users understand the rules of the language as lessons are presented. Last, we present methods and metrics for evaluating how well users are able to apply understanding of the language to create team plans which achieve specific objectives.

Portions of these contributions appear in the following works:
A. Farinelli, M. Raeissi, N. Marchi, **N. Brooks**, and P. Scerri. Interacting with team oriented plans in multi-robot systems. In *Autonomous Agents and Multi-Agent Systems*, pages 1–30. 2016.
A. Farinelli, N. Marchi, M. Raeissi, **N. Brooks**, and P. 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*, 2015.
N. Brooks, E. de Visser, T. Chabuk, E. Freedy, and P. Scerri. An approach to team programming with markups for operator interaction. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.

1.2 Reader's Guide

Below is a summary of the topics addressed in each chapter of this document. Table 1.1 summarizes the challenges addresses by each of the chapters.

Chapter 2 introduces related work in markup languages, GUI building, and team plan architectures.
Chapter 3 defines the syntax of the designed SPN and markup language.
Chapter 4 investigates mathematical properties of the SPN language derived from its Petri Net roots. Furthermore, it develops applications for these properties to assist in the creation of SPNs.
Chapter 5 presents software for developing and executing SPNs. This includes an IDE for developing SPNs using tools covered in Chapter 4. Additionally, two SPN execution GUIs are shown, one for a watercraft domain and one for an aerial domain. We discuss differences between the two domains and the resulting differences in the designed GUIs and markup interpretation.
Chapter 6 goes into further detail about the watercraft domain introduced in Chapter 5. We cover the iterative development and manufacturing cycle of two types of autonomous surface vehicles (ASV). After introducing the robot platforms, we present a set of SPN plans deployed on a fleet of these ASVs.
Chapter 7 presents an evaluation of the SPN interrupt mechanism using a simulation of the operator responding to undesired behavior with and without the interrupt sections of a SPN.
Chapter 8 presents an evaluation of the markup language using a simulation of the responsibilities of an operator and the effect on their workload and team performance when markup is applied.
Chapter 9 presents a set of tutorials developed for teaching participants with programming

experience how to write SPNs. A user study is also presented which evaluates the efficacy of the tutorials in teaching participants how to design team plans.
Chapter 10 summarizes the presented work and proposes additional opportunities to further the goals of this thesis.

Contribution	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1: SPN language	■	■	■		■		
2: SAMI language			■			■	
3: Field experience				■			
4: Training design & evaluation							■

Table 1.1: Contributions discussed in each thesis chapter

Chapter 2

Background

In this chapter we will discuss several types of related work. First we will discuss existing markup languages in various software domains. We will consider the format, role, and tradeoff between genericness and expressiveness of these markup languages. This will provide insight into how to structure a markup language for human-robot team plans which can capture useful contextual information, such as the information in Figure 2.1. Next we will discuss GUIs used to control robot teams and tools which can facilitate their development. GUI building tools provide simplified and accelerated development, lowering the barrier to customize components for specific situations. This has facilitated the design of specialized GUIs for human-robot teams in a variety of scenarios. We will sample these GUIs and consider the attributes which motivated UI design decisions. Being able to capture these types of attributes will be necessary for an effective markup language. Then we will discuss some common robotics problems with a range of algorithmic solutions. Similar to analysis of related work in GUIs, we will consider the attributes which distinguish the algorithms. These GUI and algorithm attributes will provide insight into the vocabulary needed in the markup language so that context can be described in an "actionable" manner enabling the modification of system behavior. Lastly, we will discuss existing team plan formats for coordinating groups of humans, robots, and agents. In particular, we will consider how well formats can capture the types of coordination problems we are interested in. Using insight from other markup architectures we will also consider how compatible the formats are for being "marked up."

2.1 Markup Languages

Markup languages have been used in a wide variety of domains, including text editing [117], webpage development [156, 20], data storage [22], speech synthesis [182], UI specification [71], and agent behavior [163, 107, 82].

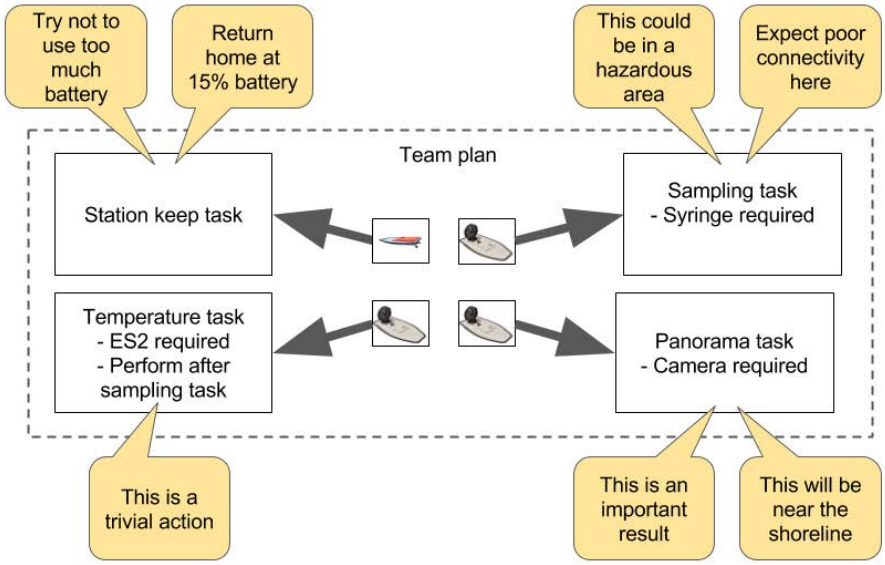


Figure 2.1: A team plan representation (dashed box) with execution context (yellow callouts)

LaTEX [117] [1] is an document preparation system commonly used for academic papers. With an extensive library of both official and user developed feature libraries, LaTEX has a vast range of capabilities. One of the main features of LaTEX that distinguishes it from traditional text editors is its markup-based approach to formatting. In traditional text editors, fonts and sizes for chapters, sections, and text must be manually specified and adjusted to improve readability and cosmetic appearance. Figures must be manually placed, scaled, and labeled. References to sections, figures, and papers must be managed manually in addition to the bibliography. Many academics have little background in typesetting, but need professional looking documents. LaTEX addresses this problem by automating these processes and others, using internal algorithms which optimize these and other aspects of the document's appearance. Instead of specifying exact fonts, kerning, and image placement, users instead add markup to their LaTEX files providing higher level information which the LaTEX engine considers when generating the document. Figure 2.2 shows some examples of the markup system being used to capture formatting, content hierarchy, and arbitrary content references at a high level. Similar to how a typical LaTEX user does not know the best practices for type setting, the typical domain expert looking to use robots in their task does not know the best human factors practices for GUIs and properties of various algorithms.

[1] `http://www.latex-project.org`

```
\section{Section 1}
\label{sec:section1}
This is the beginning of Section~\ref{sec:section1}.
\begin{itemize}
\item \textbf{Item 1 should be emphasized}
\item {\tiny Item 2 should be in a small font}
\item Item 3
\end{itemize}
```

Figure 2.2: LaTEX code

Webpages have become an integral part of modern society, from electronic commerce, to online learning, to personal blogs. A critical piece of infrastructure which allowed for this dramatic growth was the LAMP software bundle. While robust, it requires expertise to set up and maintain. As the technology has matured, the barrier to entry for webpage development has lowered in part due to development of compartmentalized languages. While there are many languages and development styles, website development tasks can be sorted into three categories:

- Designing the content of the website
- Designing the visual style of the website
- Storing and retrieving data to be displayed

As webpages have matured, these tasks have become independent of one another through the use of specialty languages for each task. Content Management Systems (CMS), such as Wordpress and Squarespace, cater specifically to non-experts interested in quickly building a website using only their knowledge of the desired structure and content. A library of "plug and play" visual style options is made available, as shown in Figure 2.5, and the necessary data manipulation architecture is hidden from the user. In an analogous system for designing human-robot team plans, the user ideally would only need to know the desired content (i.e., goals and behavior) of the plan, and would not need to modify UI code or know how to invoke specific algorithms for path planning or task allocation.

The HyperText Markup Language (HTML) [156] is commonly used to describe the structure, organization, and content of a webpage using markup in a similar manner as LaTEX. HTML specifies a number of *tags* which can be placed around text are used to indicate how content should be structured and stylistic effects, as shown in Figure 2.3. This allows content to be developed independent of knowledge or expertise about specific browser compatibilities, device resolution, or accessibility constraints.

```
<!DOCTYPE html>
<html>
<body>
<h1>First Heading</h1>
<p>A paragraph.</p>
</body>
</html>
```

Figure 2.3: HTML code

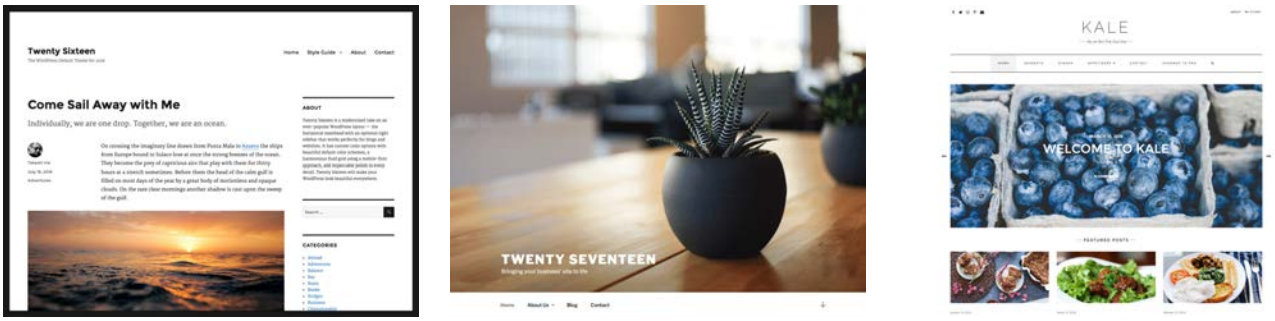
Cascading Style Sheets (CSS) [20] can then be used to interpret the markup into specific formatting commands. Margins, borders, positioning, colors, fonts, and other visual attributes are set according to the CSS interpretation, as shown in Figure 2.4. CSS allows for different interpretation depending on a number of properties including device, browser, and window resolution. In addition, the person designing the website can use existing templates designed by user interface and web design experts. This is analogous to the markup compatible UI components we seek develop which can also be reused or modified to be used across multiple application domains.

```
body {
   background-color:  lightblue;
}

h1 {
   color:  black;
   margin-left:  20px;
}
```

Figure 2.4: CSS code

A popular method for storing structured data across many domains is the Extensible Markup Language (XML) [22]. The human-readable data can be marked up hierarchically with user-defined tags describing the attributes of the underlying information, as shown in Figure 2.6. The XML data is then loaded and interpreted by an application, such as an HTML file, Excel spreadsheet, statistics program. The simple structuring of XML is one of its strengths, allowing it to easily be written, loaded, and manipulated by the most appropriate tool for the situation's demands, compared to having data embedded in an HTML file, intertwined amongst formatting and visualization code. While not a research topic in this dissertation, it is important that data collected by the robot team be stored in a similarly structured manner. The use of a common database formats such as XML



(a) "Twenty Sixteen" (b) "Twenty Seventeen" (c) "Kale"

Figure 2.5: Some "themes" offered by Wordpress

also allows for development of user-friendly webpage tools for displaying this type of data, minimizing repeated reimplementation of common tasks. Similarly, a goal of the designed language is to improve the reusability of UI components and robotics algorithms.

```
<robot>
   <id>1</id>
   <fiducial>Blue Circle</fiducial>
   <model>Lutra</model>
   <propulsion>diff-drive propeller</propulsion>
   <phone>Galaxy S3 Mini</phone>
   <date>December 2014</date>
</robot>
```

Figure 2.6: XML data

2.2 Human-Robot Interaction

2.2.1 GUI Building

Building GUIs, similar to building webpages, traditionally required expert knowledge of layout design, drawing threads, and various software patterns. GUI frameworks, such as Java Swing [2], QT [3], and GTK [4], provide support for much of this functionality, but still require a degree of expertise. As computation power and demand for easily customizable GUIs increased, the popularity of *what you see is what you get* (WYSIWYG) design increased.

[2] https://docs.oracle.com/javase/tutorial/uiswing
[3] https://www.qt.io
[4] https://www.gtk.org

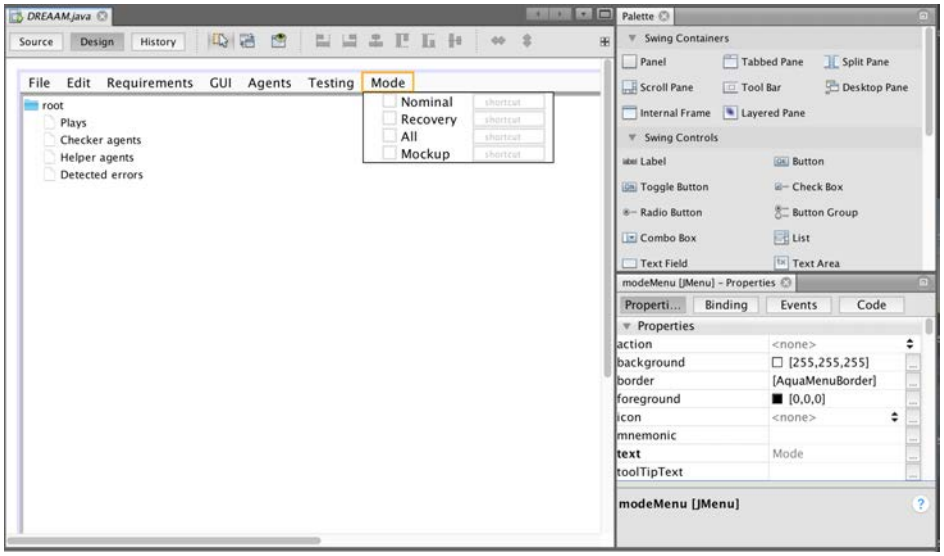
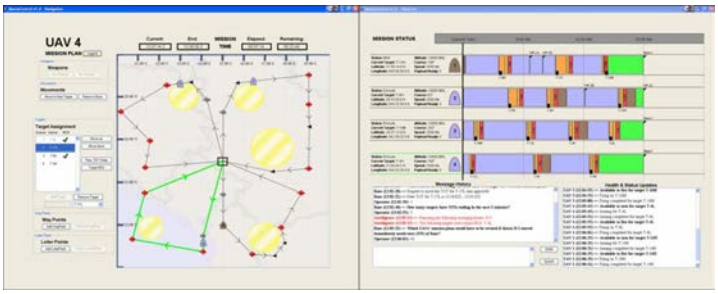


Figure 2.7: Netbeans Swing GUI Builder

In WYSIWYG GUI building tools such as Netbeans Swing GUI Builder [5], Eclipse Window Builder [6], and QT Design [7], a user drags low level components onto a grid representing the GUI, and the system constructs the underlying code to produce the visualization. These GUI building tools also automate software patterns for responding to various aspects of component manipulation. Figure 2.7 shows the Netbeans GUI builder being used to construct the menu bar for an application. The left section shows the current state of the designed GUI, the top right shows a list of available low level components, and the bottom right shows advanced options for the currently selected component in the designed GUI.

These increasingly powerful GUI frameworks and building tools have resulted in sophisticated GUIs for specific purposes [193, 199, 173, 191] and using new input modalities [134, 81, 80, 83, 100, 134, 59]. These GUIs use their domain specific context to choose strategies that best utilize the operator's cognitive resources and improve their situational awareness [87, 168, 84, 17, 3]. The MAUVE [44] GUI uses a novel "Decision Support Display" to show the schedule of events and required team resources for a team of 4 UAVs and 1 operator. The decision support display, shown on the right half of Figure 2.8a shows bottlenecks in the current schedule where operator workload will dramatically increase due to multiple assets simultaneously requiring operator resources. The decision support display provides concise visualization of the problem and can assist the operator in adjusting the schedule to reduce or avoid these bottlenecks. The SUAVE [1] GUI uses a World-In-View

[5] `https://netbeans.org/features/java/swing.html`
[6] `https://eclipse.org/windowbuilder`
[7] `http://doc.qt.io/qt-5/qtdesigner-manual.html`



(a) MAUVE UAV GUI

(b) SUAVE UAV GUI

approach to visualizing camera data from 22 UAVs in a search task. The 3D model of the world is continually "painted" using the video feeds from the UAVs and the operator can move their camera to inspect terrain from any viewpoint; Figure 2.8b shows an overhead view of the display. Chung [34] uses swarm control algorithms and two primary human roles to launch and control a fleet of 50 UAVs. The two roles are the swarm operator, responsible for selecting swarm behaviors, and the swarm monitor, responsible for checking for battery levels, altitudes, and anomalous behavior.

Several pieces of related work describe the team plan(s) within the operator GUI such that information about the operator's current task and team's objective is available to the GUI, allowing components to be modified and swapped out as needed to maximize the team's overall performance. Similar to levels of autonomy, related work approaches this topic using mixed initiative adaptation [90, 101] as well as user initiated adaptation [62, 60, 25, 118, 190]. To avoid redesigning the markup system for every domain or redesigning all GUI components for each team plan, our markup system must capture this contextual information such that team plans and GUI components exist independently.

2.3 Planning Algorithms

Similar to operator GUIs, there is a wide variety of algorithms for multi-robot planning to address the specific needs of a scenario. For path planning and motion control in multi-robot teams, there is a great deal of research [24, 13, 167, 30, 16, 183, 186, 55] which employs a variety of assumptions and techniques. Yan [198] performed a survey of multi-robot motion planning approaches and divided algorithms into 4 categories:

- Cell decomposition
- Potential field
- Voronoi diagram
- Sampling-based

The categories were compared using qualities including optimality, efficiency, safety factor, and scalable dimensionality. Yan also lists a common application for each category which

leverages its strengths, including area coverage, formation control, exploration, and manipulator control.

In the realm of multi-robot task allocation [21, 124, 45, 162, 200, 145, 130, 133, 94, 152, 116, 189], several taxonomies have been defined to analyze the extensive literature [108, 70, 198]. In Gerkey [70], algorithms were categorized using 3 primary characteristics:

- Single-task robots versus multi-task robots: Single-task robots can execute only one task at a time, while multi-task robots can execute multiple tasks simultaneously.
- Single-robot tasks versus multi-robot tasks: Single-robot tasks require exactly one robot to achieve it, while multi-robot tasks require multiple robots coordinating together.
- Instantaneous versus time-extended assignment: Instantaneous means that the available information concerning the robots, tasks, and environment does not allow for planning for future allocations, while time-extended has more information, such as the set of all tasks to be assigned or anticipated arrival times, allowing for future allocations to be considered.

Algorithms were then categorized and compared using metrics including computation required per task or iteration, communication required per task or iteration, and solution quality. Yan's [198] survey also addresses allocation approaches, and divides them by implementation category:

- Market based
- Auction based
- Trade based

The capabilities of these styles are then compared, considering factors such as utility metrics, support for task reassignment, communication complexity, and computation complexity.

Choosing the algorithm with the right balance of these competing factors requires knowledge of the context the algorithm will be used for. Categorization in literature review papers provides good insight into the types of high-level descriptors a markup language will need to capture. By including known characteristics and performance preferences for path planning and task allocation at specific points in team plans, we can then select the most appropriate algorithm from a given set of implementations.

2.3.1 Autonomy

These planning algorithms and other sources of autonomy allow robots to act independently of the operator. While this reduces operator workload for that particular decision [4, 122], operating at the highest level of autonomy is not always desirable as the algorithm may lack information only observable by the operator. In addition, removing the operator from the decision making process also lowers their situational awareness of the state of the team and task at hand [72]. One popular strategy adopted to pick the right balance between full robot autonomy and no robot autonomy is sliding autonomy [95, 194, 74, 147, 166, 51, 137, 35, 153].

System initiated sliding autonomy systems monitor the state of the system and workload of the operator and determine which decisions will be made by system autonomy and which will be made by the operator. Mixed initiative sliding autonomy allows the system to make decisions for the operator, but the operator can intervene and adjust the decision themselves at any time.

For our markup language we will grant the operator greater flexibility by allowing it to specify mixed initiative sliding autonomy settings, using the *Levels of Autonomy* [173, 149] to motivate markup descriptors.

2.4 Team Plan Representation

Now we will look at strategies in related work which defines team plan formats for combinations of agents, robots, and humans.

2.4.1 Belief Desire Intention

The key aspect of STEAM [180] is team operators, which are based on the Joint Intentions Theory introduced by [121]. In STEAM, agents can monitor the team's performance and reorganize the team based on the current situation. The TEAMCORE [154] architecture builds off of STEAM, adding the concept of *proxies* and *team-oriented programming* (TOP) to improve software integration and accelerate plan development. Each TEAMCORE agent has a corresponding proxy which serves as a middle layer between the TOP framework and domain-level agents. The proxy captures the capabilities of its agent and handles communication and coordination between other team members, and adds support for heterogeneous and distributed teams. In TOP, the programmer specifies an organization hierarchy of subteams and a graph of goals and the subteam responsible for them. The underlying TOP infrastructure then handles the formation, maintenance, and coordination of the subteams. It also allows for team reorganization in response to a team member's failure. The specific mechanics to accomplish a particular goal are left to the agent, allowing for plans to be specified at a high level independent of the actual agents which will ultimately fulfill it. The Machinetta [170] framework makes further improvements to the proxy concept, allowing for larger teams of agents to work together.

BITE [96, 97] specifies a library of social behaviors and offers different synchronization protocols that can be used interchangeably and mixed as needed. Inspired by STEAM [180], BITE also maintains a organizational hierarchy and goal behavior graph. One key addition in the BITE architecture is the introduction of a library of hierarchically linked social interaction behaviors implementing interaction protocols for synchronization and task allocation The goal behavior graph allows specifying which synchronization or task allocation algorithm is used by a particular behavior in the graph to address specific performance or robustness

needs.

While these frameworks provide methods for building team oriented plans, they do not feature mechanisms for a human operator to supervise the execution of the plans, such as directing high level objectives or providing new information. In addition, while GUIs for plan development have been created [154], the BDI architecture does not inherently provide a graphical representation of the overall plan. Furthermore, there are no inherent properties of these languages which can be leveraged to build tools for validation or verification.

2.4.2 Finite State Machines

Finite State Machines (FSM), also known as Finite State Automatons (FSA), are a popular computational model due to their visual nature and analytical properties. A deterministic finite state machine M is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of:

- A finite set of states Q
- A finite set of input symbols called the alphabet Σ
- A transition function $\delta : Q \times \Sigma \rightarrow Q$
- An initial or start state $q_0 \in Q$
- A set of accept states $F \cup Q$

However, as a FSM exists in one state at a time, it does not inherently support concurrency. To cover the state space capturing the status of each member of the team, the FSM size grows exponentially as there must be a state for every possible combination of team member's statuses. FSMs face similar difficulties modeling synchronization, and require exponential growth in the state space or additional mechanisms.

One approach to addressing these limitations is by using a hierarchy of state machines. Hierarchical finite state machines (HFSM), popularized by state charts[79], allow for better organization and reuse of state sequences in multiple contexts, potentially reducing the overall state space size. Alur [6] defines a HSM as a tuple $(K_1, ...K_n)$ of modules, where each module K_i has the following components:

- A finite set N_i of *nodes*.
- A finite set B_i of *boxes*. The sets N_i and B_i are all pairwise disjoint.
- A subset I_i of N_i, called *entry nodes*.
- A subset O_i of N_i, called *exit nodes*.
- An indexing function $Y_i : B_i \mapsto i + 1...n$ that maps each box of the i-th module to an index greater than i. If $Y_i(b) = j$, for a box b of module K_i, then b can be viewed as a reference to the definition of the module K_j. If b is a box of the module K_i with $j = Y_i(b)$, then pairs of the form (b, u) with $u \in I_j$ are the *calls* of K_i and pairs of the form (b, v) with $v \in O_j$ are the *returns* of K_i.
- An edge relation E_i consisting of pairs (u, v) where the source u is either a node of a return of K_i, and the sink v is either a node or a call of K_i.

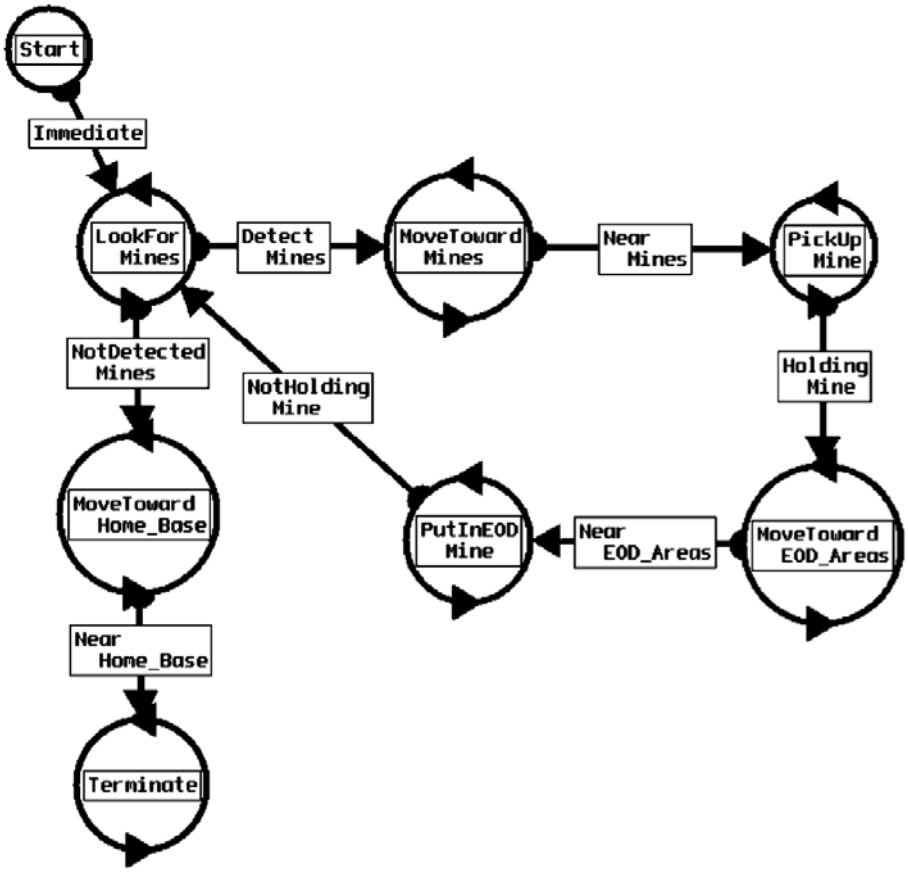


Figure 2.9: An explosive ordinance disposal (EOD) MissionLab plan (from [57]).

Marino [129] develop a HFSM-based approach for using a multi-robot team to patrol an area. The robots in this scenario are designed to be fully decentralized with no explicit communication. As no coordination between the robots is needed and the robots only need to model their own state, this allows for a compact state space. Three superstates, or boxes, corresponds to range of distances (large, medium, or small) between the robot and the perimeter of the patrol area, and the nodes, or states, inside the superstates handle the robot behavior for that condition.

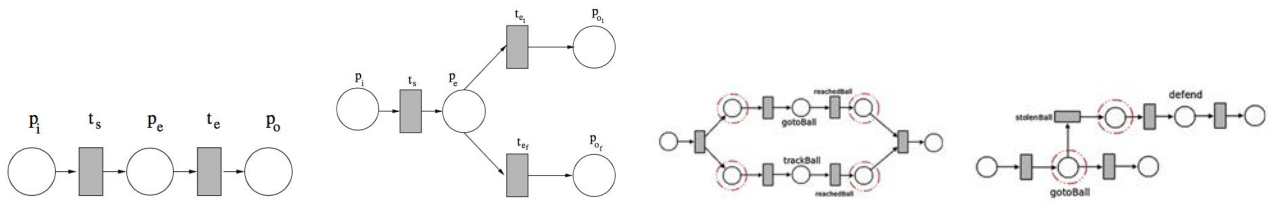
The MissionLab [8] architecture allows for mission plans to be designed using FSA in the MissionLab *Configuration Editor* GUI. States in a mission may correspond to complex tasks, which are achieved by sub FSAs. The language is used for multi-robot teams [126] where each robot has a separate HFSA modeling their own state. Limited coordination can be achieved through states where a signal is sent/received to/from a team member. Figure 2.9 shows a MissionLab plan which uses a single robot to detect and clear landmines. MissionLab features

a GUI for running missions where a human operator selects a mission to run. Once a mission starts, the operator can bias the robot teams movements to assist in exploration and can modify gains for each robot's obstacle avoidance and goal attraction. Two usability studies were conducted evaluating the usability of MissionLab by new users. In the first usability study [9], speed and accuracy of mission planning in the MissionLab configuration editor was evaluated. Participants in the study had either no programming background, programming background but no MissionLab experience, or programming background and MissionLab experience. Participants were introduced to the MissionLab language through a guide sample mission. Afterwards, the participants were provided with instructions for 5 new missions to implement independently. For a subset of the participants with programming fluency, additional experiments were conducted to compare the configuration editor to writing the mission in C code with an equivalent set of C functions as the actions in the configuration editor. In the second usability study [57], speed and accuracy of mission planning was compared between a new *wizard* and the existing configuration editor. The wizard helps users build plans by retrieving previously created missions and allowing the user to adapt them to the new situation. Participants were recruited from both technical (engineering, computer science, or math-related topics) and non-technical backgrounds and were divided into a group to use the wizard and a group to use the configuration editor. Participants completed 4 tutorials, plus 1 additional tutorial for the wizard group, before going on to 2 tests, which described a scenario to write a mission for. The resulting design time and accuracy of the created missions as well as the perceived difficulty in created them was compared.

However, as with state machines, hierarchical state machines still lack a natural way to capture the concurrent actions of multiple robots. Using separate HFSMs for each robot allows for concurrent actions in a concise manner, but limits the representation of synchronizing actions.

2.4.3 Petri Nets

Petri Nets (PN) [150], another graphical computation model, compactly support synchronization and concurrency and are a popular choice for designing, executing, and/or monitoring multi-agent processes. 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. Weights on the arcs define the number of tokens that must be present in certain places to trigger a change in the system, which results in token movement. This greatly simplifies representing the statuses of multiple team members and allows for a compact representation for synchronization. Formally, a PN is a tuple $PN = (P, T, F, W, M_0)$, where



(a) Ordinary action (b) Sensing action (c) Branching actions (d) Interrupt action

Figure 2.10: Key types of PNP structures taken from [201]

- $P = \{p_1, p_2, \ldots, p_m\}$, 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.
- $P \cap T = P \cap F = T \cap F = \emptyset$ and $P \cup T \cup F \neq \emptyset$.

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.

Petri Nets have a number of extensions [33, 61] and are a popular choice for multi-robot plans [29, 103, 110, 119, 125, 197, 11]. Petri Net Plans (PNP) [201] take 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. Different sections of a PNP correspond to activities of different robots in the team, with a token for each robot indicating its current action. The PNP is built from from PNP structures, each corresponding to some sort of action, as seen in Figure 2.10. Structures are combined in series and parallel to form complex plans.

While the use of Petri Nets allows for a centralized view of the entire team, the PNP framework includes functionality to build distributed plans for each robot from the centralized version, adding in structures for soft and hard synchronization. Another useful function offered by the formalism of PNP is the possibility to modify the execution of a plan at run-time using interrupts placed on transitions, as seen in Figure 2.10d. Task allocation is possible through PNP structures similar to sensing events, where branch options would include boolean comparators such as *closestToBall*, and *!closestToBall*.

In [40], Costelha uses Generalised Stochastic Petri Nets (GSPN) to model multi-robot plans. The framework is built around 4 types of places: *predicate* places, *action* places, *task* places, and *memory* places. Predicate places capture beliefs about a predicate and come in pairs: one for the belief the predicate is true and one for the belief that the predicate is false. Transitions and edges are created such that, for each pair of predicate places, one place

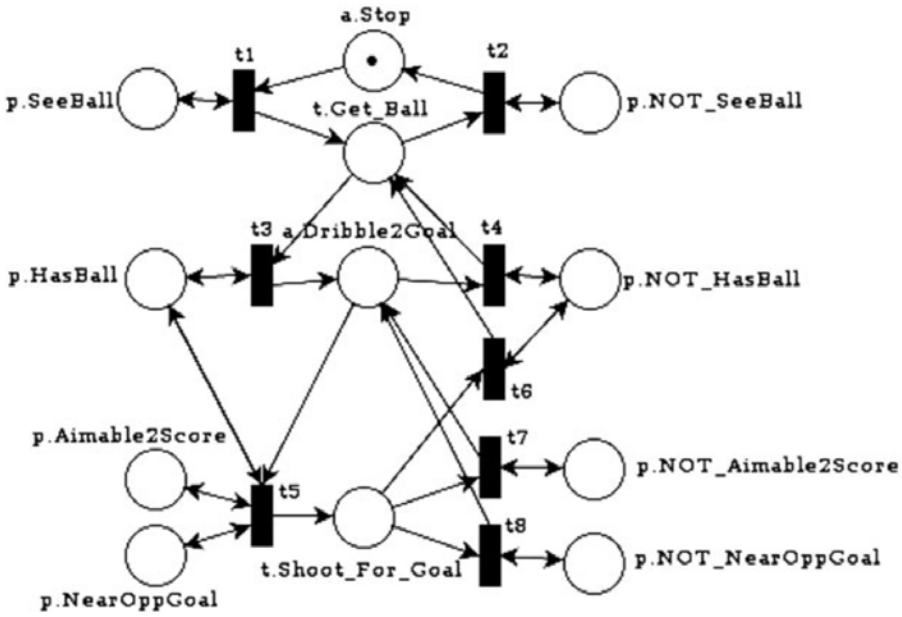


Figure 2.11: The Score_Goal task plan PN (from [40])

will have one token to indicate that is the current belief and the other place will be empty. Memory places have no specific properties and are used as a conventional PN place to mark that certain information has been obtained. Action places act as macro places [15], enabling a corresponding action PN which models that action's execution. Action PNs contain memory and predicate places, but as they are modeling atomic actions, do not contain action or task places. Task places are similar to action places in that they act as macro places enabling a task plan PN to run. Task plan PNs model a high level objective and can contain any type of place, including task places for other tasks. Figure 2.11 shows the Score_Goal task plan. The various PNs are linked together via an algorithm which merges duplicate predicate references and connects action and task macro places with their corresponding PN. Creating coordinated multi-robot teams in a soccer domain is explored in [39], where each robot has its own set of task and action PNs and synchronization between robots in those PNs is achieved using additional macro places. These macro places model the transmission (or reception) of a piece of information to (or from) a teammate, such as being ready to pass the ball (or the teammate being ready to pass the ball). The framework allows for various models of communication, from deterministic communication to communication with probabilistic transmission time and failure.

2.4.4 Colored Petri Nets

Colored Petri Nets (CPN) [91] extend Petri Nets where tokens have attached data values called the token's *color*. The firing behavior of transitions and consequently the evolution of the net's marking can depend on a token's color. In addition, transitions can modify the value(s) of the token's color when they are fired. A CPN is defined by the tuple $N = (P, T, F, \Sigma, C, N, E, G, I)$ 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 pairwise disjoint set of arcs.
- Σ is a set of color sets defined within CPN model. This set contains all possible colors, operations and functions used within CPN.
- C is a color function. It maps places in P into colors in Σ.
- N is a node function. It maps F into $(P \times T) \cup (T \times P)$.
- E is an arc expression function. It maps each arc $f \in F$ into the expression e. The input and output types of the arc expressions must correspond to the type of the nodes the arc is connected to. Use of node function and arc expression function allows multiple arcs connect the same pair of nodes with different arc expressions.
- G is a guard function. It maps each transition $t \in T$ to a guard expression g. The output of the guard expression should evaluate to Boolean value: true or false.
- I is an initialization function. It maps each place p into an initialization expression i. The initialization expression must evaluate to multiset of tokens with a color corresponding to the color of the place $C(p)$.
- $P \cap T = P \cap F = T \cap F = \emptyset$ and $P \cup T \cup F \neq \emptyset$.

Moreover, similar to PN, CPN can be analyzed and verified either by means of simulation or formal analysis methods [159], thus allowing validation of team oriented plans before their execution.

Colored Petri Net Plans (CPNP) [128] build off the basis of PNPs, but add the additional representational power of CPNs. In a CPNP extension built for multi-robot teams, tokens in CPNP represent robots, differentiated by a robot ID variable in their token color. Guards placed on transitions allow for a boolean expression using a token's variables to act as an additional firing constraint for the transition. Arc expressions allow for complex evaluations to be performed on token variables. For input arcs (arcs from a place to a transition), arc expressions limit which tokens are moved when a transition fires to those with a particular value for a variable. For example, an input arc expression can specify a particular value for the *robot ID* variable, effectively only moving that robot's corresponding token. For output arcs (arcs from a transition to place), arc expressions can change token variable values for the tokens moved by the transition's firing. Dynamic task assignment and reassignment is made possible by including variables representing the status of each task to the robot token color set. An external task allocation algorithm is then able to indicate the resulting

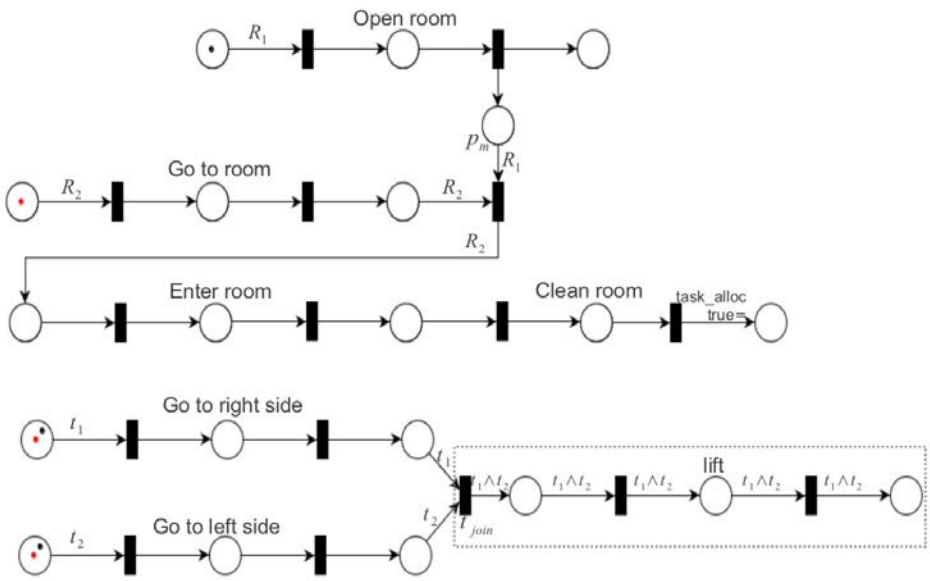


Figure 2.12: The dynamically allocated Organize Room CPNP (from [128]). Some repeated arc expressions are omitted for compactness.

allocation by changing the value of an assigned task to *in progress* in the token corresponding to the assigned robot. Figure 2.12 shows a CPNP with two parts. The top CPNP uses a predefined allocation, where robot R1 opens a room, after which R1 and R2 enter the room and clean it. After cleaning is done, task allocation is enabled via an output arc expression to allocate a joint lifting task consisting of lifting the object from its right (t_1) and its left (t_2). The bottom CPNP shows the execution of the tasks, with the allocated robots moving into position before the lift action is executed by both robots simultaneously.

2.4.5 Language Design

Visual representations have been an effective technique for teaching both teaching children [42, 43, 120] and novices [48, 50, 151, 127] to program individual robots. In addition, visual representations also provide an intuitive method for adding markup. Petri Nets offer this while adding increased representational power of sequential and parallel coordination. Furthermore, the analytical properties of Petri Nets and Colored Petri Nets could reduce the amount of training required to use the language through the use of assistance and debugging tools.

Building off this related work, we will add several other capabilities to a new team planning language based off of Colored Petri Nets. In many scenarios, especially those involving exploration, tasks will be dynamically generated. As new points of interest are discovered, further investigation may be required by a team member with different capabilities or fewer

responsibilities. Human domain knowledge is a necessary element in our applications, so operator interaction will be a fundamental aspect of the presented team planning language. While adding these capabilities, considering the implementation method's impact on learnability [105, 104, 138, 111] will be important to allow for use in real world applications where non-experts will be responsible for designing team plans.

Chapter 3

Language Syntax

3.1 SPN Definition

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 [92], which supports hierarchical CPN and the use of variables.

We define a 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.

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[1], 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 [201]: 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 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 3.1 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

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

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 ProxyPath-Completed, 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 [92], 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. Different 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 3.1) 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-mission in SPN provide a specific implementation of hierarchical CPN [92]. 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). 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 [92], 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

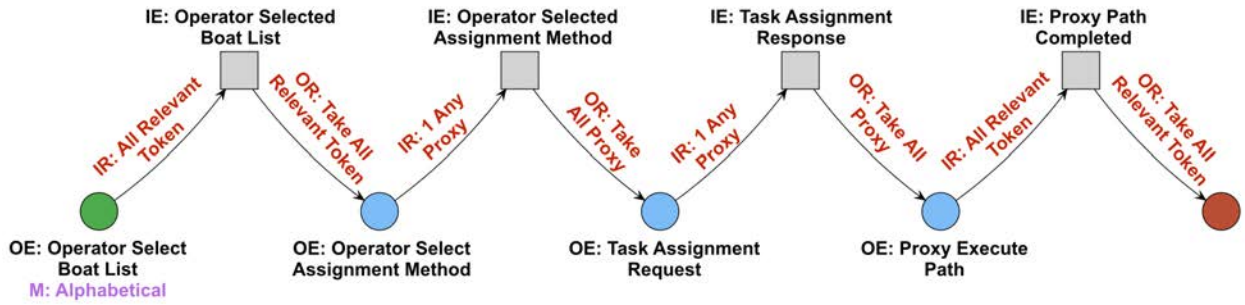


Figure 3.1: SAMI Petri Net "Cooperative Location Visit" plan (without the interrupts). The starting place is colored green and the end place is colored red

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 3.1. 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 3.2a and 3.2b. The figures display the same SPN reported in Figure 3.1. These markings illustrate how the colored tokens (related to different boats) progress through the SPN. Figure 3.2c 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.

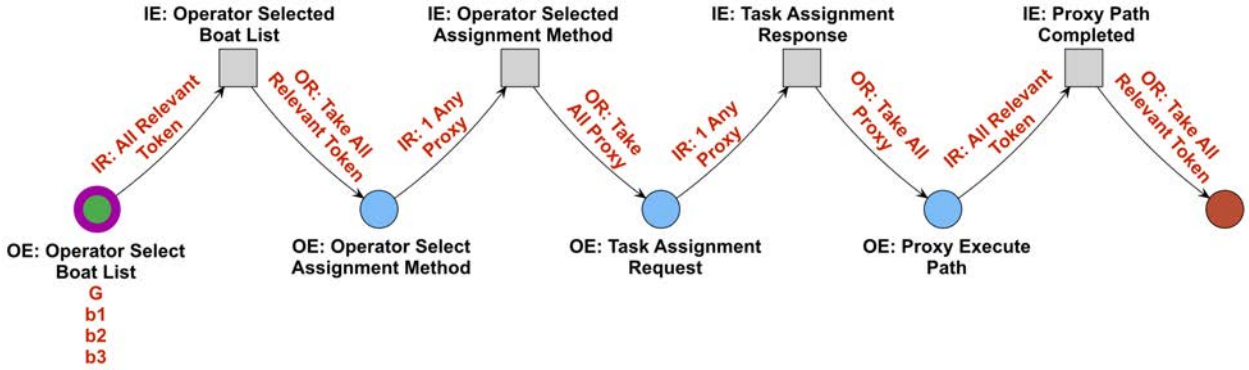
Algorithm 1 Checks if a transition should be enabled

```
procedure CHECK TRANSITION
    for ie ∈ E_I(t) do                    ▷ Check that all input events have been received
        if ie.received == false then return false
        end if
    end for
    for if ∈ t.inEdges do   ▷ Check that all incoming edge's in requirements have been satisfied
        for ir ∈ R(if) do
            if ir.satisfied == false then return false
            end if
        end for
        p = if.start
        for sm ∈ SM(p) do ▷ Check that any sub-missions on an incoming place are at a goal state
            if sm.complete == false then return false
            end if
        end for
    end for
return true
end procedure
```

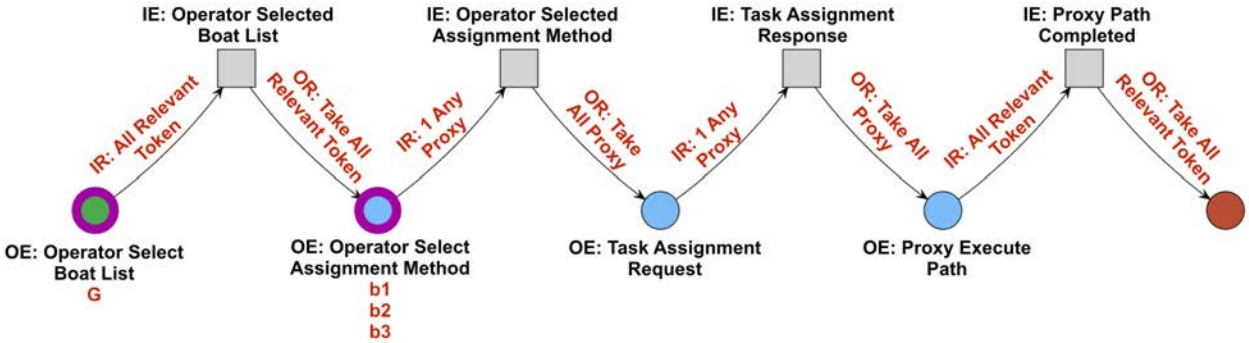
3.1.1 Summary

In this chapter, we presented the syntax rules for SAMI Petri Nets (SPN). SPNs are a significant extension to CPNs, allowing for tokens to represent robots or tasks, events to control token movement, variables for event fields, inhibitor and reset arc functionality, and nested sub-missions.

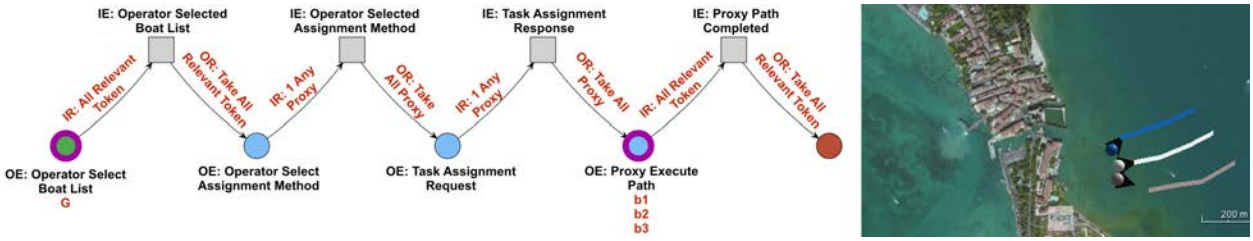
Petri Nets are not the only model which could be used to design this type of language, and there are many other ways Petri Nets could be extended to develop a similar language. Our selections were made based on our target applications at the time. One difficult language design decision was encoding behavior for an unknown team size, which could happen for a number of reasons: robots could be added or removed from the system during a plan's execution, the operator could be instructed to select a subset of the team to be used in the plan, or tasks could be dynamically created in response to a team member's actions. Manipulating an unknown group size led to the development of the "relevant token" mechanism and need for inhibitor arc support.



(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.

Figure 3.2: SAMI Petri Net showing a partial execution of the "CLV" plan (without the interrupts)

Algorithm 2 Fires an enabled transition

```
procedure FIRE TRANSITION
    t ∈ T                                   ▷ t is the transition we are executing
    TK_A = ∅    ▷ TK_A is a map associating tokens to add to outgoing places (initially empty)
    TK_R = ∅ ▷ TK_R is a map associating tokens to remove to incoming places (initially empty)
    for of ∈ t.outEdges do                  ▷ Fill in TK_A and TK_R
        for or ∈ R(of) do
            for p ∈ t.outPlaces do
                TK_A.put(p, getTokensToAdd(or))
            end for
            for p ∈ t.inPlaces do
                TK_R.put(p, getTokensToRemove(or))
            end for
        end for
    end for
    for p ∈ t.outPlaces do
        enterPlace(p, TK_A(p))
    end for
    for p ∈ t.inPlaces do
        leavePlace(p, TK_R(p))
    end for
    for p ∈ t.inPlaces do    ▷ Reset completion status of all sub-missions on incoming places
        for sm ∈ SM(p) do
            sm.complete = false
        end for
    end for
    for ie ∈ E_I(t) do       ▷ Reset receipt status of all input events on the transition
        ie.received = false
    end for
```

```
    T_check = ∅          ▷ T_check is a list of transitions we could have affected and should now check (initially empty)
    for p ∈ t.outPlaces do                                                    ▷ Fill in T_check
        for t2 ∈ p.outTransitions do
            if t2 ∉ T_check then
                t2 → T_check
            end if
        end for
    end for
    for p ∈ t.inPlaces do
        for t2 ∈ p.outTransitions do
            if t2 ∉ T_check then
                t2 → T_check
            end if
        end for
    end for
    for t2 ∈ T_check do
        if checkTransition(t2) == true then
            fireTransition(t2)
        end if
    end for
end procedure
```

Algorithm 3 Handles tokens entering a place

```
procedure enterPlace
    TK = {tk_1, tk_2, ..., tk_n}          ▷ TK is a list of tokens being added to the place
    for oe ∈ E_O(p) do
        processEvent(oe, TK)
    end for
    for sm ∈ SM(p) do
        beginSubMission(sm, TK)
    end for
    if p ∈ P_E then
        finishPlan(p, TK)
    end if
end procedure
```

Chapter 4

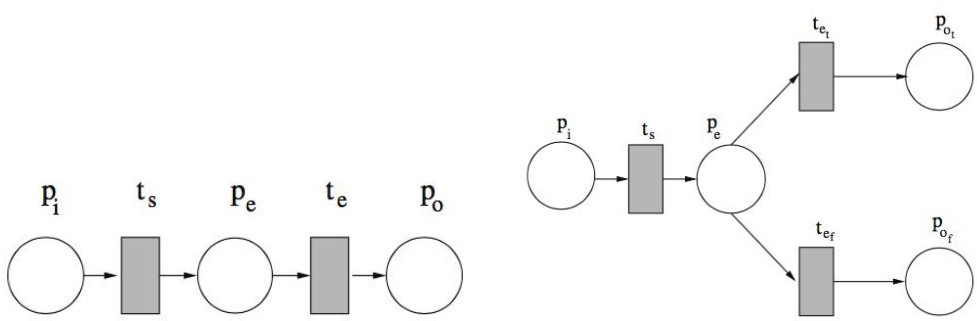
Language Properties

In this chapter, we discuss how the SPN language introduced in Chapter 3 deviates from traditional CPNs, the resulting effects on the language's behavioral properties, and methods to leverage those and other properties to assist in the development of SPNs.

4.1 Language Deviation

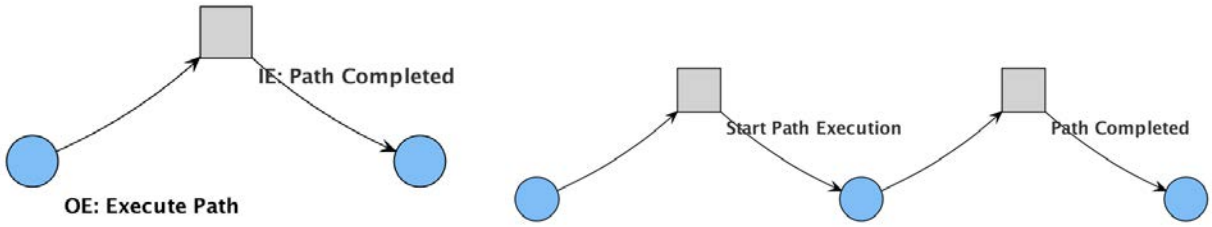
4.1.1 Events

SPNs use the concept of input and output events. PNPs [201] have a similar concept to input events in the form of *ordinary actions* and *sensing actions*. As shown in Figure 4.1, these are modeled with transitions having *events* describing the start and termination of the action. Ordinary actions model deterministic actions with an input place p_i, start event t_s, execution state p_e, termination event t_e, and output place p_o. Sensing actions model non-deterministic actions where the outcome is unknown until execution time; actions where the



(a) Ordinary action (b) Sensing action

Figure 4.1: PNP actions taken from [201]



(a) An SPN with events (b) The PNP style equivalent

Figure 4.2: Moving SPN events to only use transitions

outcome is unknown even at execution time are outside the scope of the language. Sensing actions consist of an input place p_i, start event t_s, execution state p_e, and a number of pairs of termination events $\{t_{et}, t_{ef}\}$ and output places $\{p_{ot}, p_{of}\}$. For sequential actions, the output place p_o of the first action can be merged with the input place p_i of the second action, reducing the overall size of the graph.

In SPNs, we further reduce graph size by merging p_i, t_s, and p_e into a single place, with the output event acting as both the start event and execution state. Each possible input event result for a given output event or interrupting input event are then placed on outgoing transitions t_{ie} which are analogous to termination events t_e. As shown in Figure 4.2, this merging can reduce SPN sizes by nearly half, but results in additional complexity in other aspects of the language. If we want to create a PNP equivalent to a SPN, we would begin by reversing this expansion.

4.1.2 Color Sets

In traditional CPNs, each place has an associated color: for any marking, each token in a place must be of that place's color. In SPNs, we have 3 token colors which increases representational power: Generic, Proxy, and Task. Figure 4.1.2 defines a color for each of these token types following the CPN Tools [93, 159] syntax. Knowing which type of token will be used for a given set of events and markup is difficult as some events can use multiple types and markup can further modify which types can be used. For example, the "Goto Location" output event could be activated by a proxy token or a task token. Furthermore, a place with a "Operator Approve" output event could be activated with a generic token, but if "Relevant Proxy" markup is added, then the place would need to be activated with proxy or task tokens to make use of the markup. As a result, we treat each token and assign each place's color to Task_token, as both Generic_token and Proxy_token are subsets of Task_token. Arc expressions can be used to differentiate between these token types to conform to edge requirement rules given the following implementation rules:

- Generic tokens: $P_id == T_id == Undefined_id == 0$
- Proxy tokens: $P_id > 0$ and $T_id == Undefined_id == 0$
- Task tokens: $T_id > 0$

```
Declarations:
color P_id = int;
color T_id = int;
color T_class = string;
color Generic_token = φ;
color Proxy_token = product P_id;
color Task_token = product P_id * T_id * T_class;
var Undefined_id = 0;
```

Figure 4.3: Color declarations for SPN token types

4.1.3 Out Edge Requirement Actions

Edge requirements in SPNs have substantial differences from standard PNs resulting from the use of output events to model both the initiation and ongoing execution of an action. Consider a scenario where the battery drops below a threshold, resulting in a "Low Battery" input event being generated. Depending on the context of different parts of the SPN, each place may handle the event differently or ignore it. Sections near end states may ignore the warning, while other sections may want to perform additional computation before deciding whether to substitute another robot. In the latter case, we would want the robot to continue its current responsibilities while events on other places and transitions do the required computation and selection. These computations will likely require use of the robot or task's token and, in a traditional PN structure, would require moving the token out of the current place to the new place which marks the beginning of the computation section, p_c. In a PNP, to prevent interrupting the action being executed in p_e, we would need to loop a token back to the place in addition to adding it to p_c. In a SPN model with the combined states, this would result in entering p_{oe} again, which would trigger both the initialization and execution of the action and not just the execution action. To address this while maintaining the benefits of the smaller PN using the merged p_{oe} places, the following modifications to edge requirement behavior were introduced.

For a PN, in edge requirements list tokens that must be present in the connected place for the connected transition to fire. If an enabled transition fires, the listed tokens are removed from the associated place. In an SPN, when an enabled transition fires, the listed tokens are not removed.

(a) An SPN consuming a required token

(b) The PN equivalent

Figure 4.4: Conversion of "Consume" out edges in an SPN

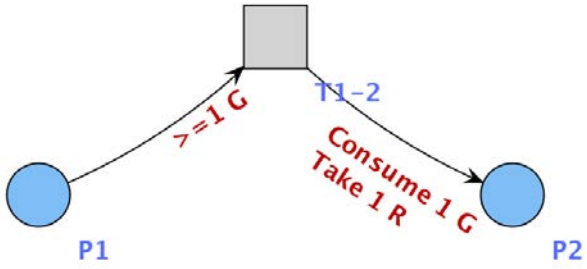
For a PN, out edge requirements list tokens that should be added to the connected place when the connected transition fires. In a SPN, when an enabled transition fires, out edge requirements can choose from 3 different actions that can be taken for the listed tokens: consume, add, or take. We will now investigate how to translate this behavior to that of a standard CPN.

Consuming Tokens

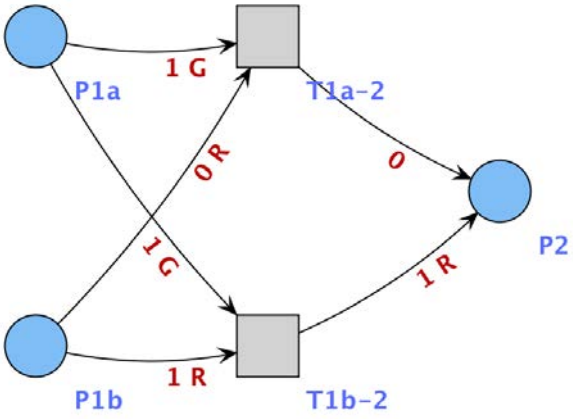
For Consume out edge requirements, the SPN manager attempts to remove the listed tokens from all the connected transitions's incoming places. When the listed tokens are also the listed tokens for the in edge, this can easily be translated to a PN, as shown in Figure 4.4. However, if the tokens are not also specified in an in edge, no standard CPN equivalent can be constructed. There are two ways the marking of the SPN in Figure 4.5a could change when the transition fires. If P1 has both a Generic and Robot token, both will be removed from P1 and the Robot token will be added to P2. If P1 has a Generic token but no Robot tokens, the Generic token will be removed from P1. This cannot be modeled using a separate transition for each of these results, because there is no guarantee that when a Generic and Robot token are present that the first transition will fire, as both transitions would be enabled. This would require the concept of transition *priority*, where a transition cannot fire if a transition with higher priority is enabled. One method for enforcing priority is via *inhibitor arcs*, an extension to Petri Net which allow in edges inhibit a transition from firing if the connected place has any tokens. Figure 4.5 shows a CPN equivalent of the SPN using an inhibitor arc.

Adding Tokens

For Add out edge requirements, the SPN manager adds the listed tokens to the connected place places, as in a typical CPN. However, to convert an SPN "add" out edge to a CPN, additional out edges would be needed to add tokens to the transition's incoming place, as the SPN in edge did not remove them. Figure 4.6 shows a SPN using generic tokens and a CPN equivalent.

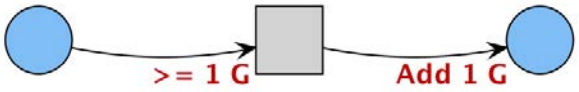


(a) An SPN with more out edge requirements than in edge requirements

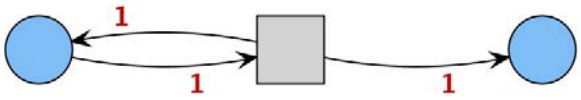


(b) The CPN equivalent

Figure 4.5: Conversion of "Consume" out edges in an SPN

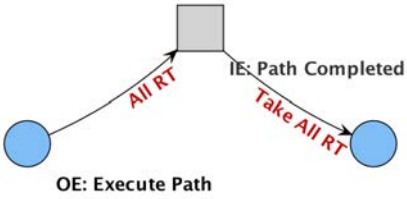


(a) An SPN adding a required token

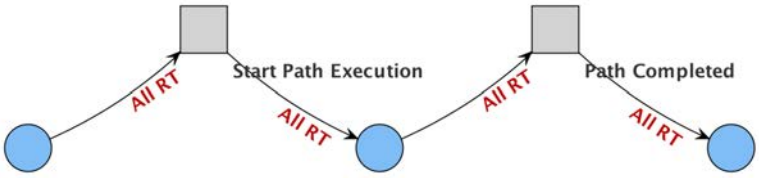


(b) The PN equivalent

Figure 4.6: Conversion of "Add" out edges in an SPN

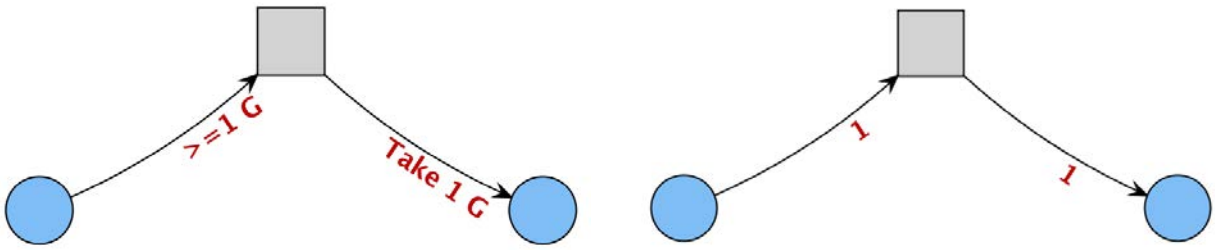


(a) An SPN with events and edge requirements



(b) The CPN equivalent

Figure 4.7: Conversion of an SPN with events and edge requirements



(a) An SPN taking a required token

(b) The PN equivalent

Figure 4.8: Conversion of "Take" out edges in an SPN

Taking Tokens

For Take out edge requirements, the engine attempts to remove the listed tokens from all the connected transitions's incoming places and add the listed tokens to the connected place. It is equivalent to the combined effects of a Consume and Add out edge requirement for the listed tokens. When the listed tokens are also the listed tokens for the in edge, this can easily be translated to a PN, as shown in Figure 4.8. When there they are not listed in the in edge requirements, a concept of priority must be used similar to Figure 4.5.

4.1.4 In Edge Requirement Quantity

In standard PN and CPNs, in edge requirements specify the minimum quantity of a token necessary for the connected transition to be enabled. This corresponds to the Greater Than or Equal To "$\geq$" quantity in SPNs. However, in many scenarios this is not sufficient to captured desired behavior, in particular when the robot team size varies or tasks are created dynamically in the plan. For example, consider a scenario where a group of robots is selected by the operator to execute a formation path. After the entire formation has reached the destination, another action should be taken. If the size of the group is known, we know the number of tokens that should have been moved once the entire formation has finished and could add edge requirements accordingly. When the group size is unknown, this is not possible. In another scenario, the operator may provide a list of locations, and a task is created for each one. Barring complications, the SPN should not terminate until all tasks are completed. If the number of tasks created was known, tasks could be moved into a common place upon completion and the number of tasks could be used by an edge requirement for the transition to an end place. When the number of tasks is unknown, this is not possible.

Less Than

To address this limitation, SPN in edge requirements can instead specify the maximum quantity of a token for the connected transition to be enabled using the "Less Than" quantity. Specifying "Less Than 1" for a type of token prevents the connected transition from firing if any tokens of that type are present in the connected place. This is equivalent to the zero testing inhibitor arc extension to PN and CPNs. A "Less Than" requirement with quantity larger than 1 is equivalent to a threshold testing inhibitor arc.

Relevant Tokens

As individual or sets of tokens enter places, they activate the output events attached to the place. These output events in turn send commands to team members, such as the operator, robots, or AI services. These team members in turn return input events as they complete the received commands. The tokens that should be moved as a result of the input event varies depending on the event. For instance, plans may commonly use the operator to choose a subset of the robot team for a particular goal or recovery action. This is achieved using the output event "→Operator Select Robots" and input event "←Operator Selected Robots". When the operator makes their selection and the "←Operator Selected Robots" is generated, only the tokens corresponding to the selected robots should be moved into the next place. The other tokens should remain in the original place. In another scenario, a set of sensor measuring tasks has been created. When a robot is assigned the task, the SPN directs it to move to its associated location and afterwards perform an action, such as recording a sensor measurement. This movement is achieved with the output event "→Proxy Execute Path" and input event "←Proxy Path Completed". When the SPN executes, it is possible multiple robots will be moving to their assigned task's location simultaneously. When a robot reaches its task's location, it generates a "←Proxy Path Completed". A mechanism is needed so that only the token representing that robot's task is moved when the transition fires.

In a SPN, the tokens to be moved in these examples can be referred to as "relevant tokens" in edge requirements. In a CPN, this concept could be represented by using multiset arc inscriptions to capture the number of relevant tokens and adding guard to the transition limiting token movement to a generic token or specific proxy or task tokens.

All Tokens

In other circumstances with a dynamic number of robots or tasks, the plan may require a transition to move all of the corresponding tokens. For example, when a set of tasks are dynamically created, the SPN developer may want to display some information on the GUI before beginning task execution. If the developer wishes to move the task tokens through the place(s) and transition(s) used to display this information using a list of tokens, this would require knowing the number of tasks ahead of time. To address this, out edge

requirements can specify that "All" tokens of a particular type present in the incoming place(s) be consumed, added, or taken. The "All" expression can be translated to a CPN multiset arc expression, and the type of token moved can be controlled by using guards on the transition, similar to Relevant Tokens handling.

4.1.5 Sub-missions

Sub-missions provide a simple mechanism to reuse common sequences of actions and compartmentalize sections of SPNs. A root mission may have any number of sub-missions, and sub-missions may have sub-missions of their own. While sub-missions do not add representational power, they can speed up development and increase readability. When a parent SPN spawns a sub-mission, the sub-mission behaves like a root mission with the following exceptions:

- Sub-missions may have a different initial marking than if it was spawned as a root mission
- Sub-missions have additional variable namespaces to consider (all parent mission namespaces)
- Parent SPNs can generate events for their sub-missions; for instance, if a parent mission is aborted, it will generate abort input events for its sub-missions

For the parent SPN, sub-missions can have the following effects:

- Sub-mission completion is required for a transition to be enabled
- Sub-mission tokens can be added via an out edge requirement if the connected transition has any in places which have a sub-mission

Sub-missions can be represented in a PNP format in a similar fashion as output and input events (Section 4.1.1), with transitions for the start sub-mission event and a sub-mission termination event. In SPNs where sub-mission tokens are added into the parent mission via an out edge requirement, a CPN equivalent can be constructed using arc expressions and guards, similar to Relevant Tokens handling.

4.2 Behavioral Analysis

4.2.1 Reachability Graphs

One of the advantages of using a PN model is the ability to analyze several behavioral properties [28, 158, 28, 53] to find syntax and semantic errors in a developed plan.

Reachability graphs are one method for performing this analysis [46, 192, 7]. These graphs take an initial marking M_0 and construct the resulting state space of the PN by iteratively expanding enabled transitions. Each node in the graph represents a unique marking

which may be reached by more than one firing sequence. Each edge in the graph represents the enabled transition which has been selected to fire. One limitation to this approach is Petri Nets with unbounded places: places which may have an infinite number of tokens as the result of some firing sequence. This occurs when the following are true:

$$\exists\, sequence \mid M_a \xrightarrow{\text{sequence}} M_b$$
$$M_b \geq M_a \iff \forall p \in P,\, M_b(p) \geq M_a(p)$$
$$\exists\, p_u nbounded \in P \mid M_b(p) > M_a(p)$$

This sequence can trigger an infinite number of times, generating a unique marking each time, which would thus require an infinitely large reachability graph. Support for unbounded places is added in coverability graphs.

4.2.2 Coverability Graphs

Coverability graphs [161] address this problem by identifying these infinite firing sequences and using ω to represent unbounded quantities in a node. For every marking $M'' \neq M'$ on a path from the initial marking M_0 to M' , if $M' \geq M''$, then $M'(p)$ is set to ω for all $p \in P$ with $M'(p) > M''(p)$. Note that, if there are no unbounded places, the reachability graph and coverability graph are identical.

Inhibitor Arcs

An important property of Petri Nets is monotonicity, which guarantees that adding tokens to a marking will not decrease the number of enabled transitions. Given a PN with markings M_a and M_b, we formally define this property as follows:

$$M_a \geq M_b \Rightarrow \forall t \in T \mid enabled(M_b, t) \Rightarrow enabled(M_a, t)$$

Petri nets with inhibitor arcs, or PTI nets, complicate behavioral analysis as they violate this property [27]. Monotonicity is an important property for collapsing the state space in the reachability graph, as unbounded places can be simplified to a single coverability graph node using *omega*. However, in recent work Reinhardt [160] provides a method for determining reachability for PTI nets with a single inhibitor arc. Van de Nes [188] provides a method for constructing a coverability graph for PTI nets with a single inhibitor arc. Methods for constructing graphs for nets with multiple inhibitor arcs are explored, but require some knowledge of the boundedness of the inhibitor place. Fortunately, in practice SPNs typically use only a single inhibitor arc in order to trigger the termination of a plan with a dynamic number of tasks. For scenarios in which multiple inhibitor arcs are needed, they will likely be bound by the team size or number of generated tasks: in this case, analysis may be possible for "supported" team and task pool sizes.

4.3 Graph Based Assistants

If a reachability or coverability graph can be constructed, properties of the graph can be used to create syntax and semantic tools, referred to as "Assistants."

4.3.1 Reachability

Given a reachability or coverability graph for a PN, a marking M_b is *reachable* from marking M_a if there exists a path in the reachability graph from M_a to M_b. This is particularly useful to see if goal markings are possible from the initial marking M_0. Conversely, it can be used to test that no path exists between two states.

4.3.2 Boundedness

A place is called *k-bounded* if it does not contain more than k tokens in any reachable marking, including the initial marking. Petri nets in which all places have a k-boundedness of 1 are referred to as being *safe*. For a PN with a finite reachability graph, it can be called k-bounded if all of its places are k-bounded. Unbounded PNs have at least one unbounded place.

As SPNs use tokens to represent team members and tasks, there are many situations where safeness would be violated due to expected behavior. Instead, the desired boundedness may instead be the size of the team or number of tasks. Analyzing boundedness for a given team or task size could identify syntax causing unnecessary duplication of tokens which could result in unexpected behavior or difficulty understanding the state of the team at runtime.

4.3.3 Other Assistants

Several other assistants can be used without a reachability or coverability graph. However, some of these assistants require "meta-knowledge" about events. For output events, this knowledge consists of the types of tokens it uses (e.g. generic, proxy, task, and/or relevant tokens) and the input events it can generate. For input events, the knowledge captures the types of tokens it acts on and the types of tokens it produces.

Start and End Places

This assistant checks that the root of the coverability graph is a start place and that it is the only start place in the SPN. In addition, it checks that there is at least one end place in the SPN and provides a warning otherwise. If any end place exists as a non-leaf in the graph, an error is generated.

Events

Given meta-knowledge of events, errors are generated if, for an input event on a transition, there is no incoming place with an output event which can generate that input event. Conversely, an warning is generated if, for an output event on a place, there is no outgoing transition containing that input event.

Token Requirements

Edges without edge requirements are immediately flagged as an errors. Given meta-knowledge, errors are also generated if, for an output event, no incoming transition's out edge requirement would provide a useable token type. Similarly, warnings are generated if a transition's in edge requirements or out edge requirements do not act on token types relevant to its input event(s).

4.3.4 Summary

In this chapter we discussed properties of the SPN language which could be used to assist in team plan development. The developed graph based assistants required special properties of SPNs to be mapped back to CPN extensions, allowing standard analytical properties to be used. Non-graph based assistants provide additional assistance based on syntax rules of the language and meta knowledge of the events used in the domain.

The focus during language development was making SPNs as user friendly as possible, not to preserve analytical properties. Depending on the target demographic's background, the balance between initial complexity and increased analytical power could be shifted.

Chapter 5

GUI

In this chapter we will present an IDE for designing SPNs and SAMI compatible operator interfaces.

5.1 DREAMM IDE

Figure 5.1 shows the developed Integrated Development Environment (IDE) for SPNs, named DREAMM. [1]

Mission Tree

The Mission Tree on the left side of the GUI shows the list of SPNs and sub-missions in the loaded project file. Users can reorder, rename, delete, and clone SPNs via this panel or select the mission that is shown in the Mission Model View.

Mission Model View

The Mission Model View in the center of the GUI is used to view and edit the selected SPN. Developers can use the mouse to reposition or delete individual or groups of vertices and mouse menus to make changes to a vertex's label, events, and markup.

For plans with several contingencies and interrupts, the large number of edges can make it difficult to gauge connectivity. Specifically, it complicates viewing the incoming places for a transition which would trigger a contingency or interrupt. To address this, two different visualization modes were created - nominal and recovery. *Nominal* mode shows all of the details of the SPN, as seen in Figure 5.2a. *Recovery* mode shows only the silhouette of the graph's vertices, hiding all text and all labels as shown in Figure 5.2b. If a vertex or edge

[1] Other collaborators in DREAMM development included Athena Johnson, Dan Bernstein, and Richard Johnson

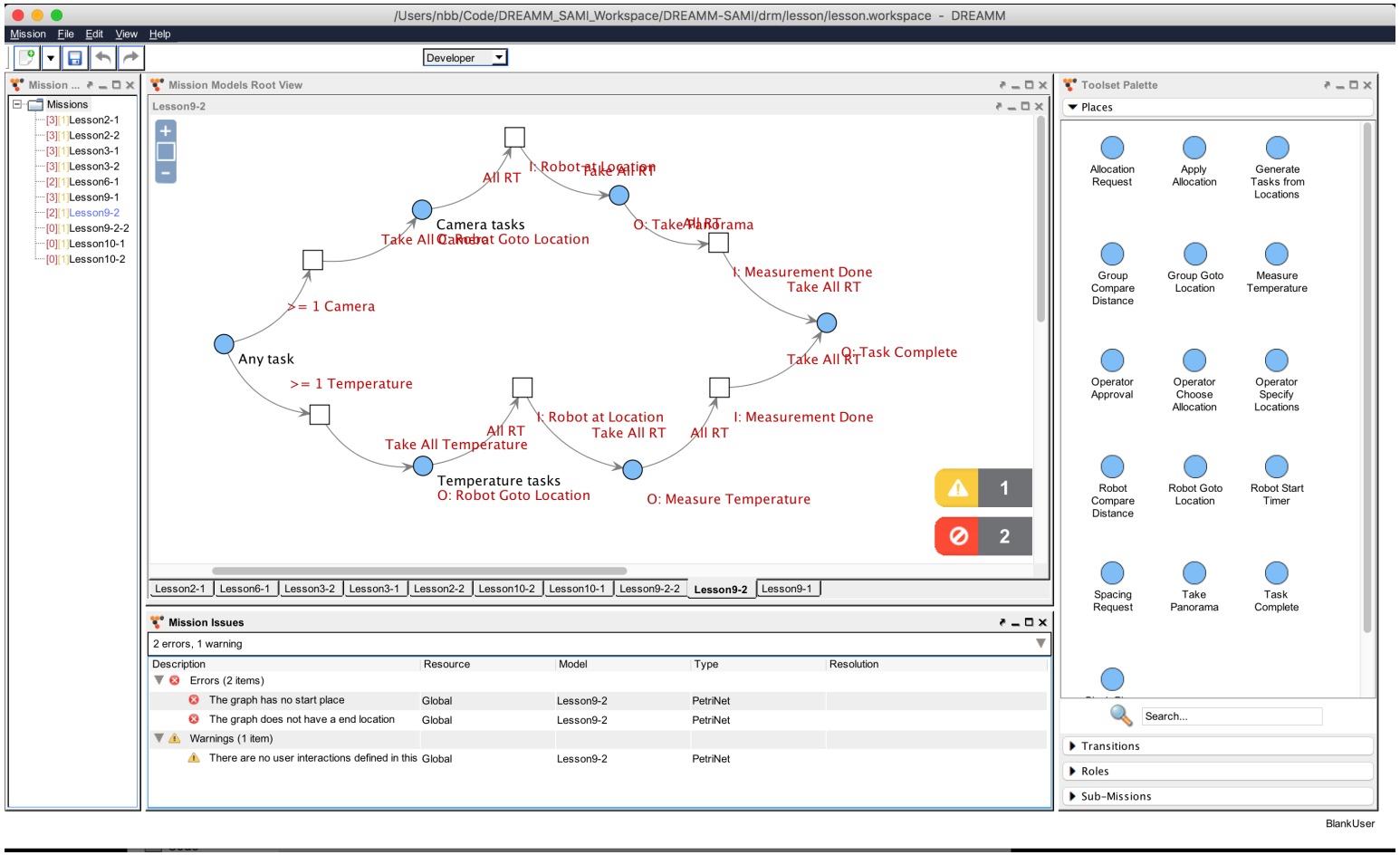


Figure 5.1: DREAMM SPN editor

is created while in recovery mode, it is referred to as a recovery vertex or edge. Recovery vertices and edges are hidden when the IDE is in nominal mode, but are otherwise identical to vertices and edges created in nominal mode. When a vertex is selected while in recovery mode, a subset of the graph is fully displayed, showing the selected vertex's connections to other nominal and recovery vertices. This allows the developer to quickly check which sections of a plan a particular contingency or interrupt covers and how it could affect SPN execution. Figure 5.2c shows the result of selecting a transition while in recovery mode which has one incoming place and one outgoing place. Figure 5.2d shows the result of selecting a recovery transition while in recovery mode which is connected to every place in the SPN.

Figure 5.3 shows the event editor window used to edit settings and variables used by events on a vertex. The left third of the frame contains a text searchable list of available events. The middle third of the frame shows the current events on the vertex with options to delete them or reset their settings. The right third of the frame shows the variable specification and definition options for the event selected in the middle frame.

Toolset Palette

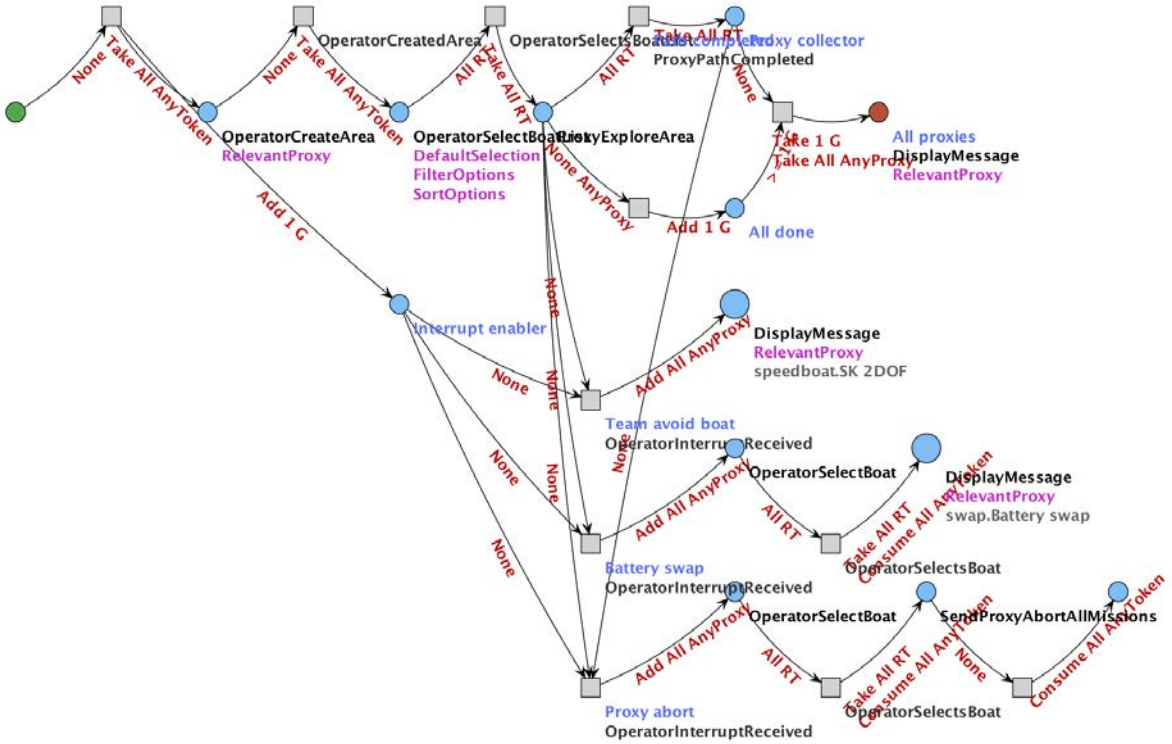
The Toolset Palette on the right side of the GUI allows developers to locate specific events and sub-missions which can then be dragged and dropped to be added into the SPN. Dropping onto an existing place or transition will add the element to it (if syntactically allowed) and dropping onto a blank space creates the appropriate place or transition and adds the element to it.

Mission Issues View

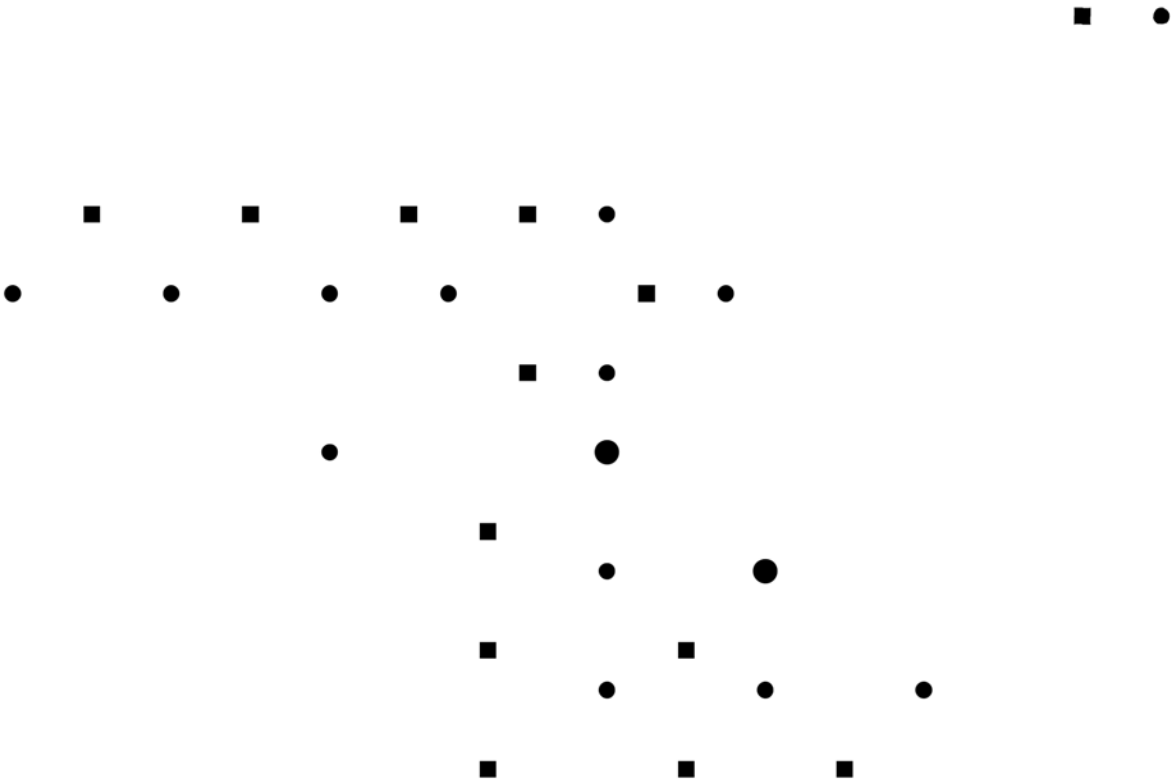
The Mission Issues View in the bottom center of the GUI displays any *errors* or *warnings* that have been detected. The bottom right of the Mission Model View displays the number of each of these for quick reference. Errors are illegal or missing syntax which will prevent the SPN from being executed. Warnings do not prevent a SPN from executing, but may result in undesired behavior. Chapter 4 discusses the SPN properties which are used for this process. Each entry in the table has a text summary of the problem and possible solution. When an entry is selected, any places, transitions, or edges associated with the entry are highlighted in the Mission Model View.

5.1.1 Event Wizard

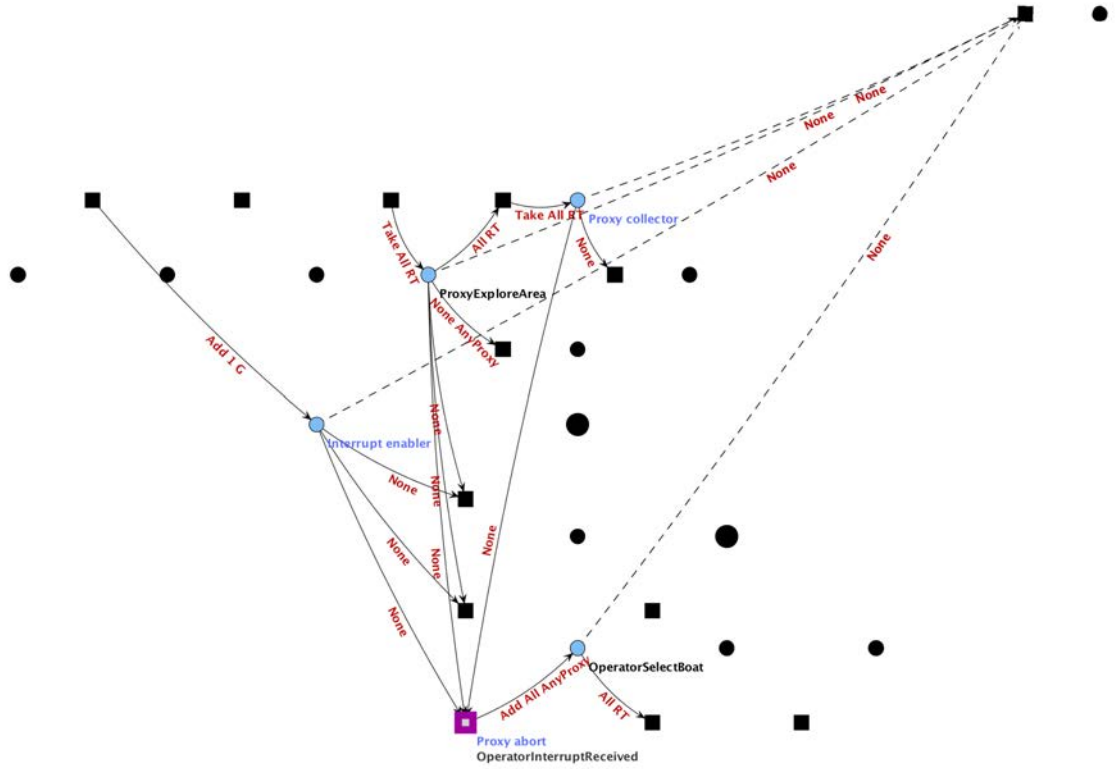
Developing plans in DREAMM can require performing many repetitive actions to add in simple, predictable functionality. For instance, after dropping "ProxyStartTimer" from the Toolset Palette, "ProxyTimerExpired" is dropped, a second place is created, and edges and edge requirements are added to them. To reduce these repetitive actions, an "Event Wizard"



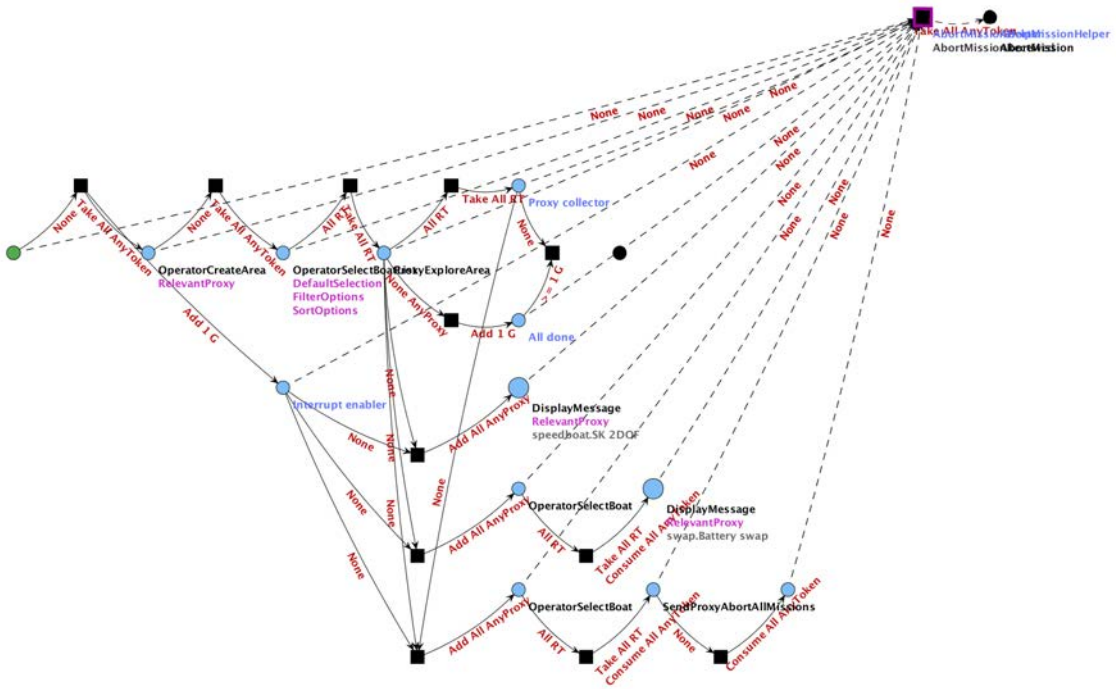
(a) Nominal mode



(b) Recovery mode, no vertex selected



(c) Recovery mode, vertex selected



(d) Recovery mode, vertex selected

Figure 5.2: Using the recovery visualization mode in DREAMM

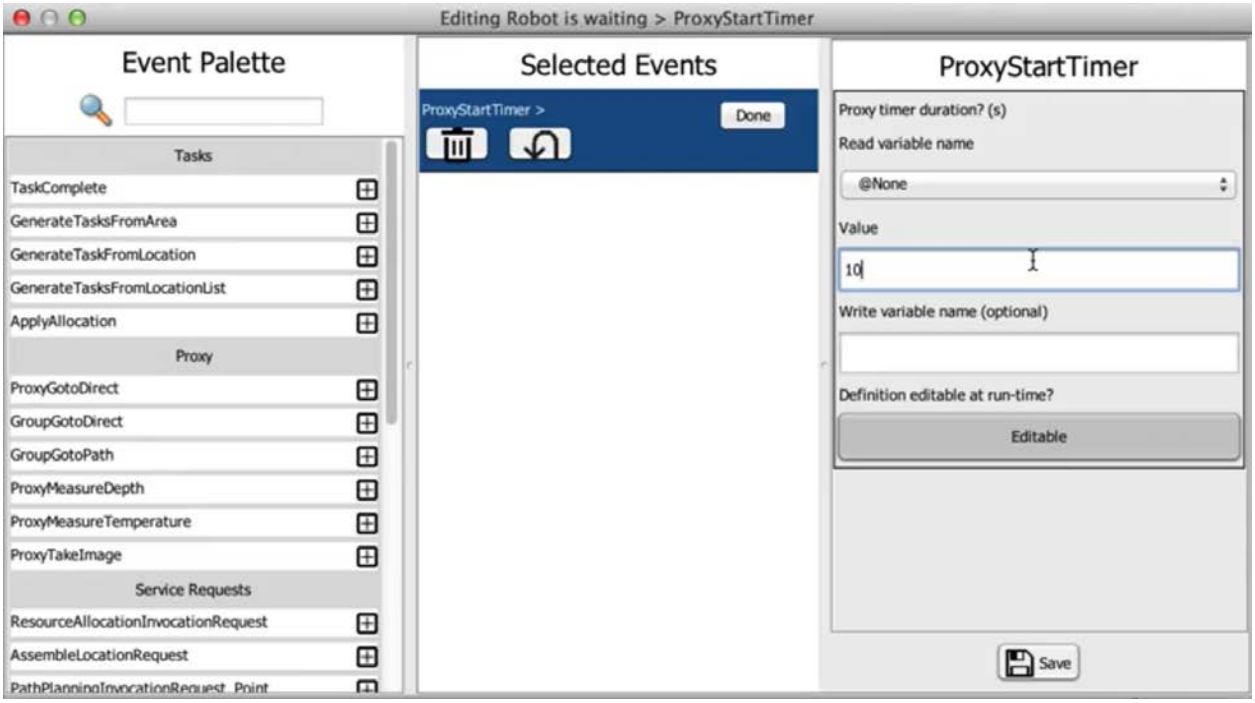


Figure 5.3: Modifying variable for an output event

was created which could take a representation of a sequence of actions to be performed after dropping an event and automate its execution. Also developed was an edge requirement computation algorithm, which would assist in adjusting edge requirements in the overall plan as events were added.

Sequence representations were divided into two categories: *flexible* and *manual*.

A flexible sequence for an output event defines a list of input events that should lead out of the dropped event and the types of tokens (none, proxy, or task) the event needs to execute. For example, ProxyStartTimer would have an input event list containing ProxyTimerExpired and would require a proxy token to be used. For each input event listed, a new transition leading out of the output event's place is created and the input event is added to the transition.

A flexible sequence for an input event defines a list of output events that should lead out of the dropped event and the minimum in edge requirements and out edge requirements necessary for the event to perform as expected. For example, ProxyTimerExpired would have an empty output event list and require an "All Relevant Tokens" in edge requirement and a "Take All Relevant Tokens" out edge requirement.

A manual sequences for an event explicitly defines all the vertices, events, edges, and edge requirements, which are then instantiated in the SPN. Manual sequences are used when a

flexible sequence cannot capture necessary requirements. For example, a flexible sequence cannot specify that tokens should be removed from a place as an input event occurs and that a transition connected to a second place should be enabled when the first is empty. Importantly, the edge requirement computation algorithm cannot modify edges created by a manual sequence.

If an event is dropped and there is no flexible or manual sequence for it, it is added to the SPN and the requirement computation algorithm is triggered. The algorithm assumes that event does not require any tokens to operate.

Edge Requirement Computation

Each time an event is added or deleted from the SPN by the developer or wizard, a edge requirement computation algorithm attempts to update edge requirements as needed. Consider the following example: A developer starts a new SPN by dropping a "SystemStartTimer" output event. It has a flexible sequence definition listing the "SystemTimerExpired" input event and not requiring any specific tokens. "SystemTimerExpired" has a flexible definition listing no output events and listing no in or out edge requirements. As no specific tokens are required, the computation results in using a generic token to activate "SystemStartTimer" and "SystemTimerExpired" is given the in and out edge requirements "$\geq$ 1 Generic" and "Take 1 Generic," respectively.

Next, the developer drops a "OperatorSelectRobot" output event onto the outgoing place created for the "SystemTimerExpired" transition. "OperatorSelectRobot" has a flexible definition listing the "OperatorSelectedRobot" input event and requiring proxy tokens. "OperatorSelectedRobot" has a flexible definition listing no output events and listing "All Relevant Tokens" as a in edge requirement and "Take All Relevant Tokens" as an out edge requirement. These edge requirements are added, as well as a "$\geq$ 1 Generic" and "Take 1 Generic," so that the generic token is moved forward through the plan. Additionally, to ensure the proxy token requirement for "OperatorSelectRobot" is fulfilled, "Take All Proxy" is added as an out edge requirement for "SystemTimerExpired."

5.2 GUI

In this section we present two GUIs designed for different domains which are compatible with SAMI markup. Figure 5.4 shows the architecture of the execution software, separated by various knowledge requirements.

5.2.1 CRW GUI

Figure 5.5 shows the GUI designed for the Cooperative Robotic Watercraft (CRW) project.

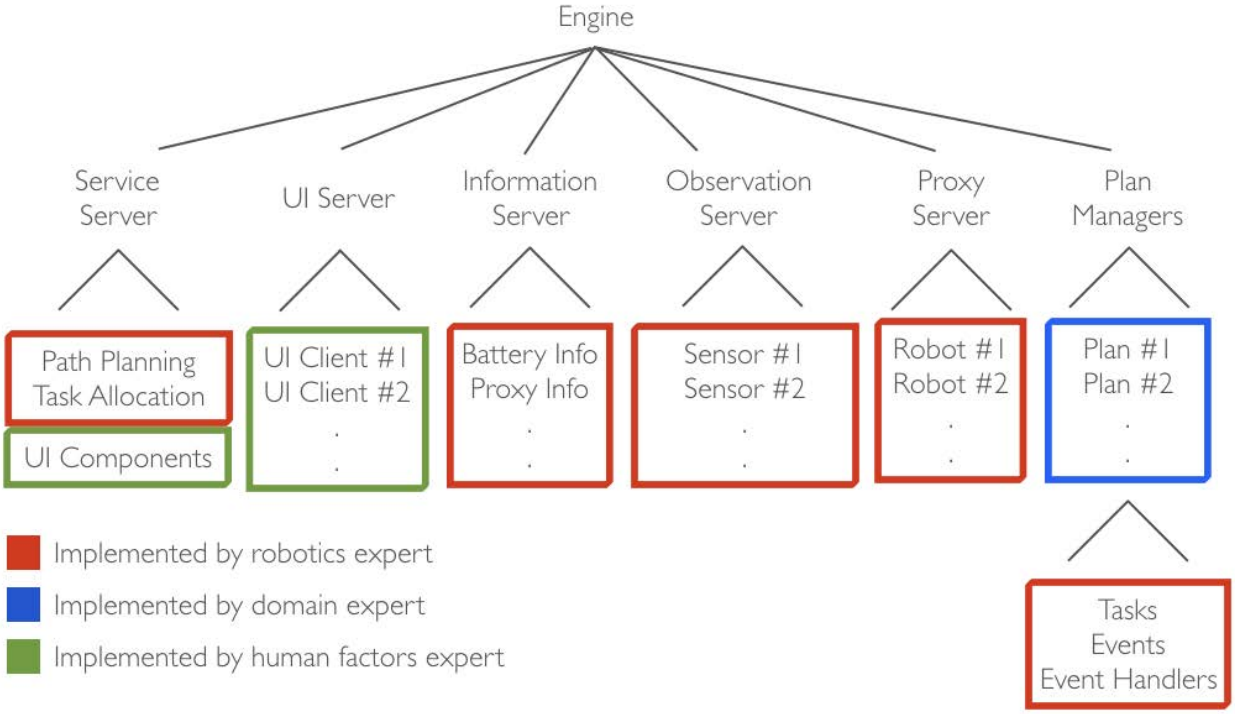


Figure 5.4: SAMI system architecture design

Mission Monitor

SPNs are invoked and monitored in the top-right section of the GUI, using the same SPN visualization format SPN developers are familiar with.

The left third of this GUI is used to display information using the Message Frame, Communication Frame, Interrupt Frame, Allocation Frame, and Map Frame.

Message Frame

The message frame handles text based information sent to the operator by SPNs.

Communication Frame

The communication frame provides a summary of the status of each team-member, using color coding and text as necessary by SPN markup.

Interrupt Frame

The interrupt frame is used by the operator to trigger interruptions in the currently running SPNS.

Map Frame

The map frame shows a map, the team members, and their currently executing action. If necessary, the operator can manually manipulate which sensor overlays are shown. The map frame also contains buttons for activating low-level control of a robot, including teleoperation, which overrides commands sent by SPNs in the event of dangerous, unexpected behavior. The decision queue in the bottom-right section of the GUI handles all SPN interactions with the operator. Rather than manipulating the components on the left third of the GUI to present text options or request definitions on the map, Algorithm 4 is used to construct a component for each interaction. As SPNs generate interactions, they are added to a priority queue, using Priority markup on interactions to sort the queue. The top interaction in the queue is then shown in the bottom-right of the GUI. In Figure 5.5 the only interaction in the queue is specifying the area for a team of boats to map.

5.2.2 AIMS GUI

Figure 5.6 shows the GUI designed for Adaptive Interface Management System for Netcentric Supervisory Control of Multiple UAVs (AIMS). [2] The components on the left border of the

[2]Other collaborators in AIMS GUI development included Athena Johnson, Dan Bernstein, and Richard Johnson

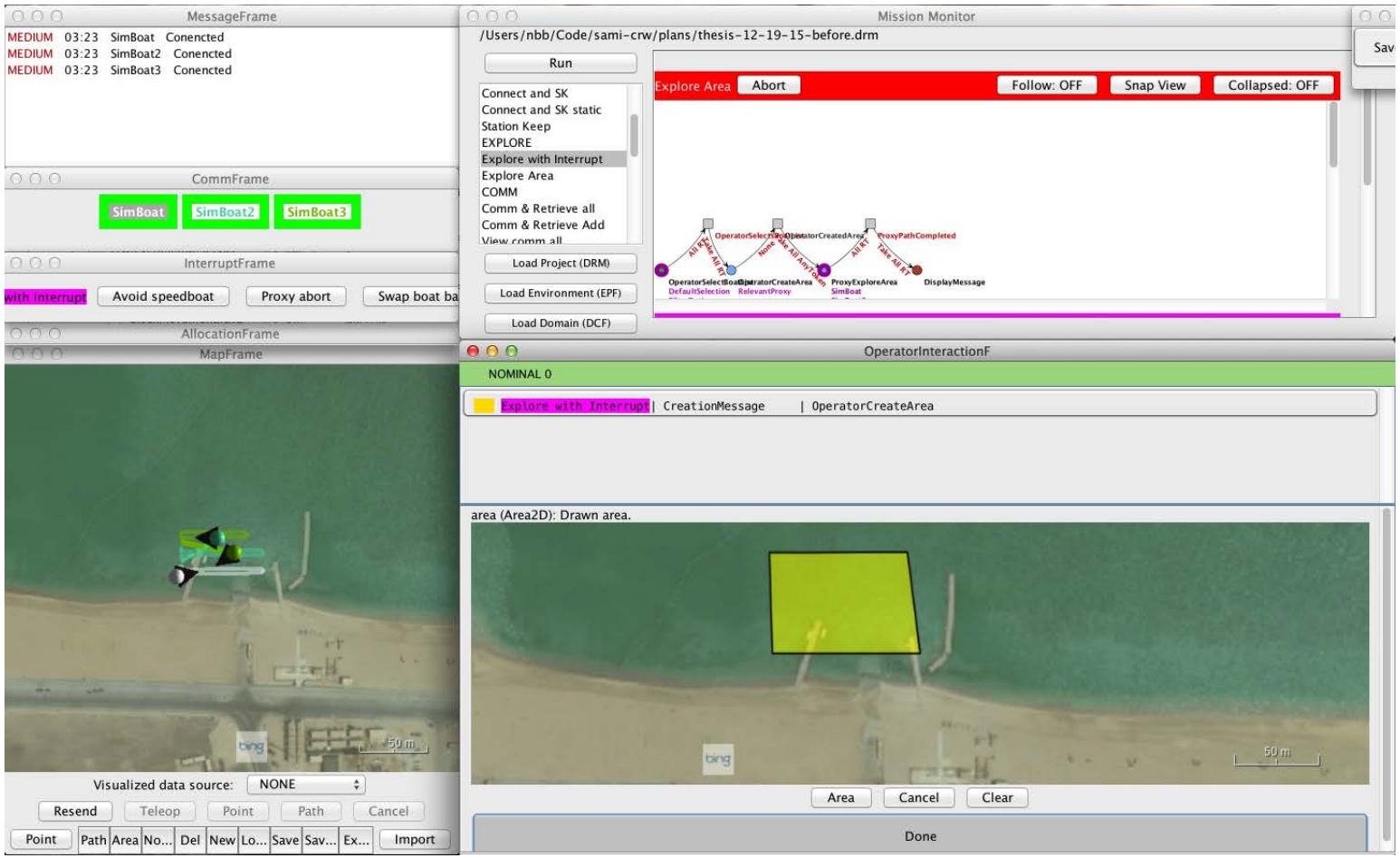


Figure 5.5: CRW GUI

GUI are used to summarize the status of assets in the team and show the most relevant video feed.

The top-right component is a map used for visualizing, selecting, and creating geographic information.

The bottom-right component is a message box similar to the message frame in the CRW GUI.

The bottom-center component is the mission manager, which has 4 tabs: Plan, Gantt, Detect, and Decisions. The Plan tab allows the operator to view descriptions of available SPNs and start instances. The Gantt tab shows a Gantt chart inspired visualization for each running SPN called the phase chart, which is discussed in more detail below. The Detect tab, analogous to the interrupt frame in the CRW GUI, is used to trigger interrupt behavior in SPNs. The Decision tab is similar to the decision queue in the CRW GUI and shows the top decision in the priority queue. However, instead of creating a new map component for geographic decisions, it modifies the existing map. In Figure 5.6, the current decision is approving paths for a set of UAVs.

5.2.3 Phase Chart

The Phase Chart was designed to provide the information shown in the SPN visualization, but in a format understandable to operators with limited exposure to Petri Nets. Figure 5.7 shows the phase chart for a SPN describing a search operation, which has been divided into three sections, or *phases*: getting locations of interest from the operator, assigning search tasks at these locations to team members, and performing the search tasks. The first row of the chart contains sequentially ordered buttons for each "phase". Clicking one of these buttons displays the sequence of actions which take place in that phase of the SPN in the subsequent rows. In Figure 5.7, the "Task assignment" phase has been selected and its actions are displayed. Each of the subsequent rows corresponds to a team member, such as the operator, the SAMI engine, or a UAV. Actions are placed in the row corresponding to the team member responsible for executing it. Each teal box represents an action in that phase which has not yet been executed. When an action is executed, the box changes from teal to green. Operators can preview future branches in the plan by clicking on the different branching actions and the chart will update to reflect the effects of that branching action. In Figure 5.7, the effects of the "Yes" branching option for the "Operator Chooses Allocation" is being previewed. The purple box indicates the end of the current phase and contains the name of the phase SPN execution will continue in. In Figure 5.7, the purple box indicates that the "Task assignment" phase has no more actions and that future actions continue in the "Task execution" phase.

The chart is built when a SPN is started by searching through it for two types of markup: Phase Item and Phase Branch. By incorporating the phase structure into the SPN using markup, we can use all the SPN's knowledge about sequential and parallel properties of

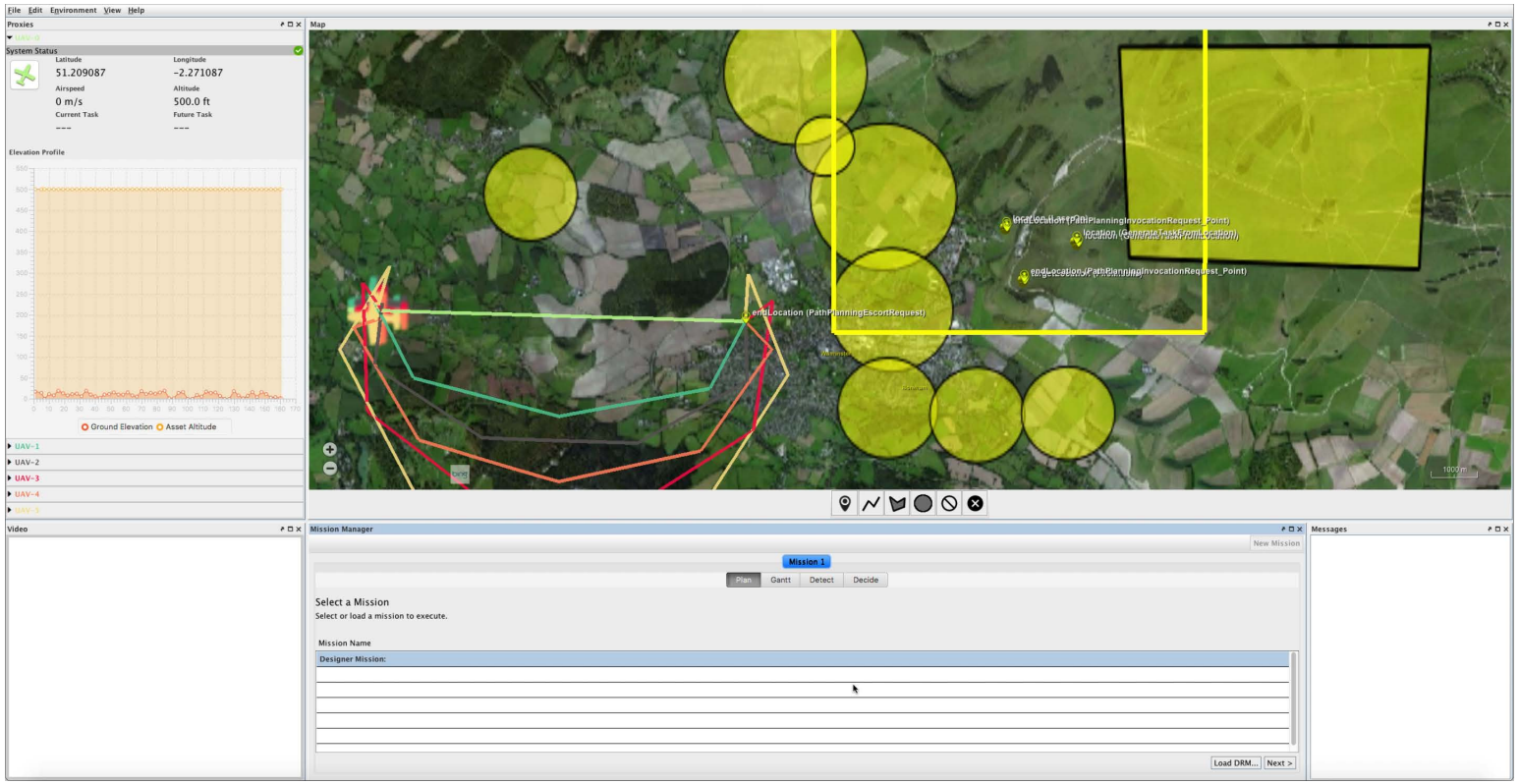


Figure 5.6: AIMS GUI

actions when building the chart.

Adding PhaseItem markup on an event indicates that an action box should be added to the phase chart for the associated place or transition. Phase item markup contains two pieces of information: the name of the phase it belongs to and the team member responsible for it. The phase name is used to determine which phase the action box should be added to. The responsible team member determines which row the action box is placed in. Team member options include operator, SAMI, Proxy, and user-named tasks.

Adding PhaseBranch markup on an event indicates that a branching option should be added to the previous PhaseItem action box in the phase chart. This adds a node as described in PhaseItem and also adds a button onto the node for the nearest PhaseItem (going backwards in the SPNs connectivity graph). Phase Branch markup requires the same information as Phase Item plus a short description of the choice which would lead to this branch's execution. This text is used to label the button added to the action box for this branch option.

In the following examples, we copy the information specified in a phase related markup into the place or transition's label. If the place or transition has Phase Item markup, the label will have two sets of brackets: the first set of brackets contains the *Phase name* and the second set the *Actor responsible*. If the place or transition has Phase Branch markup, the label will have a third set of brackets containing the *Action name*. Figure 5.8 shows the SPN section and resulting phase chart for the "Get locations" phase of the plan. Figure 5.9 shows the SPN section and resulting phase chart for the "Task assignment" phase of the plan. Figure 5.10 shows the SPN section and resulting phase chart for the "Task execution" phase of the plan.

5.2.4 Component Construction

Components and widgets

To increase code reusability, UI elements are often separated into *components* and *widgets*. Components are stand alone components, such as maps and text boxes. Each component has a set of widgets, which are optionally added to an instance of a component to add additional complexity, such as a map layer showing planned paths for robots. Widgets may reduce the operator's context switching time for certain scenarios by compactly presenting certain types of information, but may increase it in others when that information is unnecessary and the widget obfuscates important information. Requiring operators to manually adjust component and widget settings as needed is also time consuming. To address this, many UIs have rules where widgets are activated, deactivated, or modified as the operator enters different modes of operation. For instance, when teleoperating a robot, additional video feeds for the robot may be shown or enlarged. Markup allows developers to take this further, by specifying contextual information which affects the ideal configuration. One way to provide this ideal configuration is to specify a configuration of components and widgets

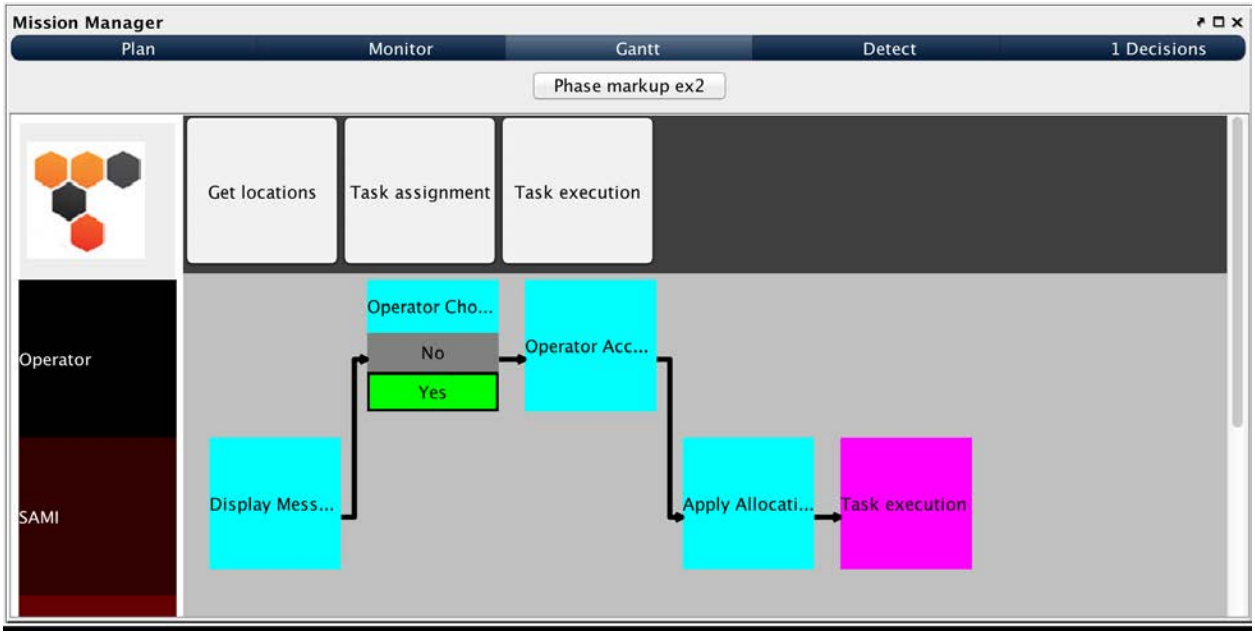


Figure 5.7: Phase markup for UAV search plan

for every combination of interaction data type and markup set. However, this becomes increasingly difficult as the numbers of interaction types and markup options grow, resulting in exponential growth for hand-coded UI component and widget configurations and clear motivation for automating this process [67, 102]. We address this problem by dynamically creating and modifying UI components to conform to the requirements of an interaction's data types and markup.

Component capabilities

We divide operator interactions in SPNs into two categories, *creation* and *selection* decisions. Creation decisions are those in which some sort of definition must be provided by the operator, such as a point of interest. Selection decisions are those in which information is presented to the operator and may require the operator to make a choice, such as selecting from a set of path planning solutions for a robot. Displaying information is considered a subset of selection decisions, where no actual selection needs to be made.

We refer to markup compatible components and widgets as markup components and markup widgets. Each of these has several data structures capturing what data types they can be used to create or select as well as what markup options they support. Using this knowledge, we can automatically construct the minimum set of UI components and widgets necessary to create or select information of a certain data type and markup set. For existing components, we can modify widget usage to use the minimum set necessary for a particular

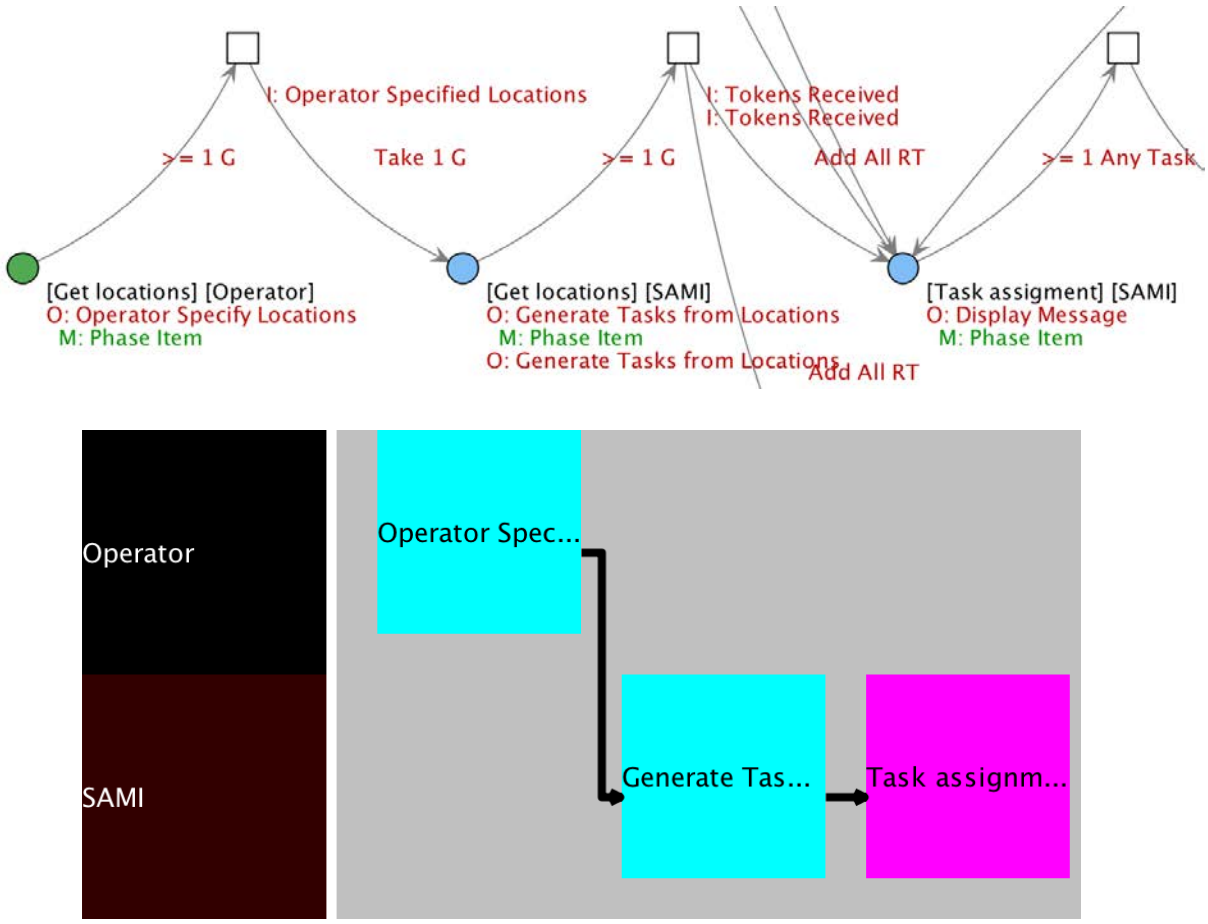


Figure 5.8: "Get locations" SPN section and phase chart

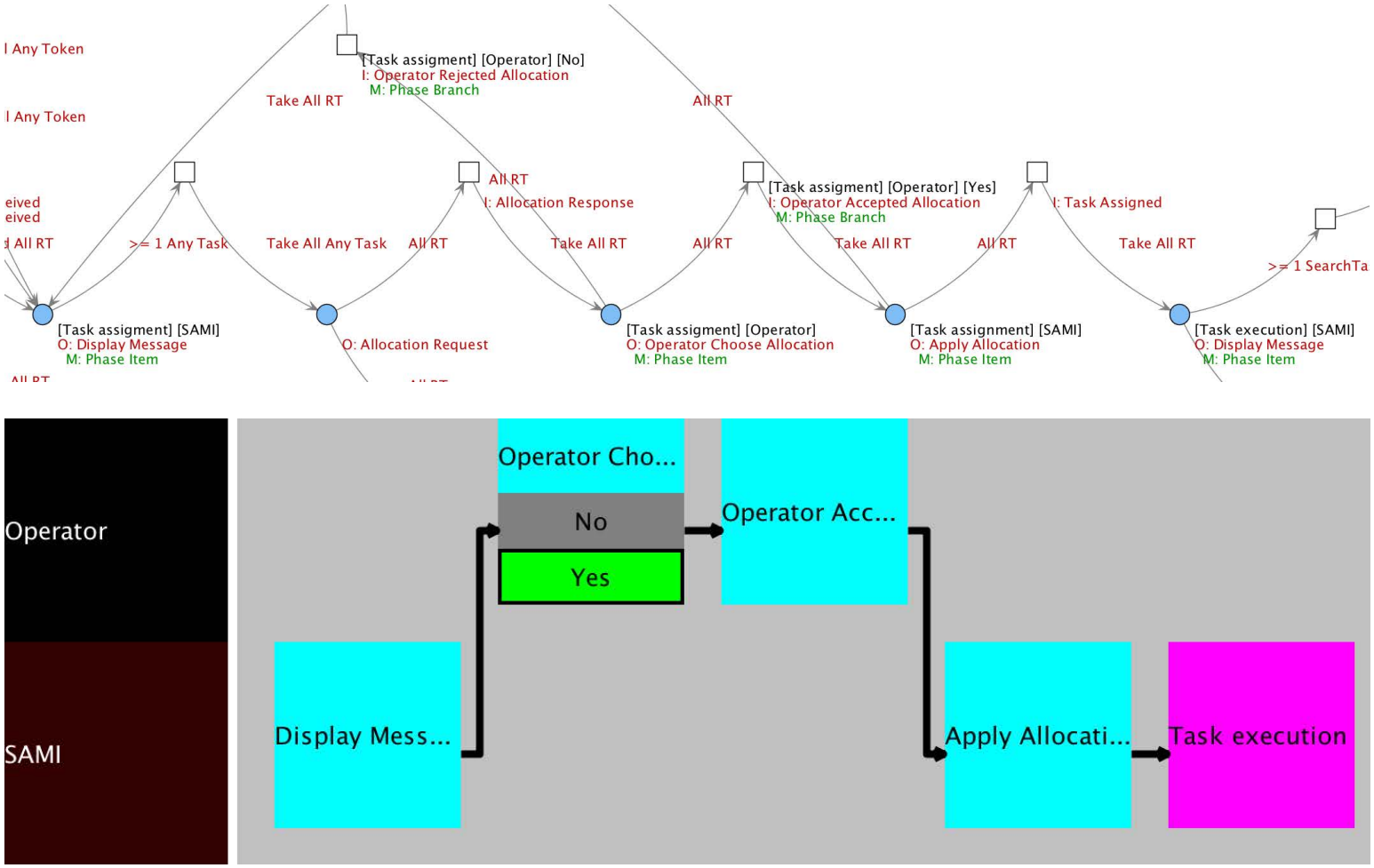


Figure 5.9: “Task assignment” SPN section and phase chart

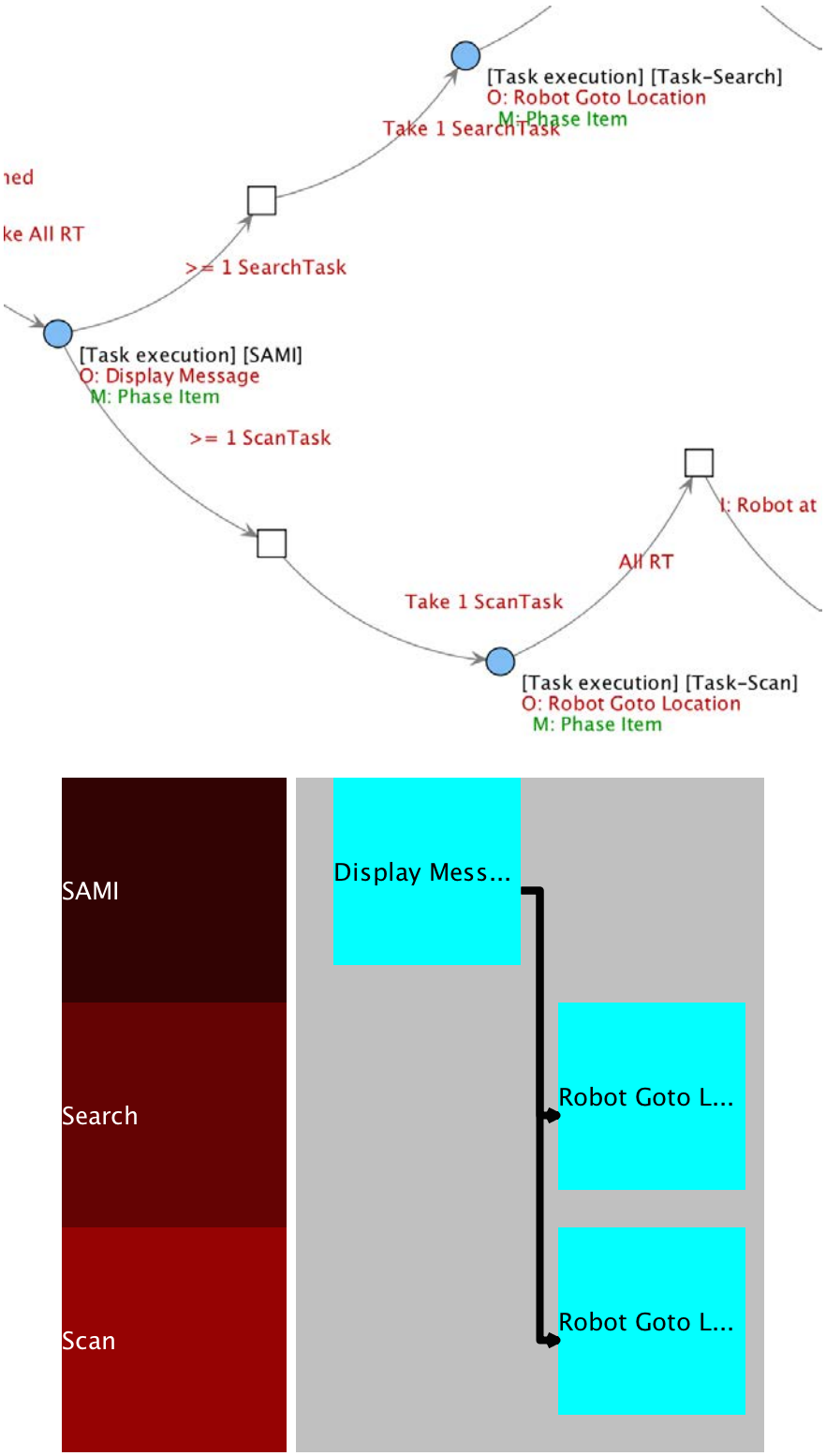


Figure 5.10: "Task execution" SPN section and phase chart

decision. This functionality enables increased reuse of code within a database and could allow for simpler integration of UI research from external sources [41, 178, 175].

Dynamically constructing components also provides benefits when using custom data classes. As SPN functionality expands, often custom data classes are required to capture a set of information composed of existing data classes. Traditionally, new components would have to be constructed for each of these custom data classes, though the new component may simply be a list of the existing components for each sub-class. We can automate this by recursing into data types and attempting to find a set of components that collectively represent a data type when no single component can.

Each markup component and widget defines the following lists:
SCC (List<Class> singleCreationClasses): This can be used to create an single instance of these classes
SSC List<Class> singleSelectionClasses): This can be used to select from/show a single instance of these classes
MCC (List<Class> multiCreationClasses): This can be used to create a list of instances of these classes
MSC (List<Class> multiSelectionClasses): This can be used to select from/show a list of instances of these classes
HCC (Hashtable<Class, List<Class>> hashtableCreationClasses): This can be used to create a list of table entries for these class pairs
HSC (Hashtable<Class, List<Class>> hashtableSelectionClasses): This can be used to select from/show a list of table entries for these class pairs
MO (List<Enum> markupOptions): This can be used to interpret these markup options

In addition, markup components define the following:
WC (List<Class> widgetClasses): Markup widget classes which can be attached to this component class

This automation is achieved in two parts: scoring and construction. Scoring determines if creating a component that captures the full scope of the decision is possible and, if so, to what degree the component will support the specified markup. Construction uses the scoring to create or modify the instances of the components and widgets.

Algorithm 4 is the overall scoring algorithm. For each quantity case, it initially uses Algorithm 5 to try to find a single component and widget set for the decision. If this is unsuccessful, the next action varies based on the quantity. In the single quantity instance, it attempts to recursively find a collection of components which capture the decision using Algorithm 6. In the case of a multiple quantity, such as a list of a class, the algorithm searches for a component which supports a single quantity of the class. This component can be cloned as many times as necessary to show definitions (in the case of selection) or

create definitions (in the case of creation). If this fails, then recursion is used for find a set of components which can show a single definition, which is similarly cloned as many times as necessary. In the case of a hashtable quantity, the algorithm searches for a component for the hashtable key's class and another component for the hashtable value's class. If this fails for either/both class(es), it recursively searches for a combination of components for the failed class(es). The returned value is -1 if no collection of components can capture the interaction's data type.

Algorithm 5 uses the data structures mentioned above to try and find a single component and widget set which captures the data structure and maximizes markup support. To preserve space and clarity, only a single case is handled. Other cases follow the same pattern, but use the corresponding list. If a component or one of its widgets supports the data type, then the component's score is set to 0. The score is incremented for each markup option which the component or one of its widgets supports, with the max score possible equal to the size of the markup list. The best score keeps track of which component supports the data type and supports the most markup options. If none of the components and widgets support the data type, the returned value is -1.

Algorithm 6 is invoked by Algorithm 4 when Algorithm 5 fails to find a single component to represent a data structure. This algorithm breaks down the provided class, or key and value classes, into a list of their subclass (line 5). Algorithm 5 is then used to find a component for each of these subclasses. If that is not possible, the algorithm recurses into that subclass until either a supporting component is found or a recursion limit is reached. If the recursive search for a component reaches the recursion limit at any point, the algorithm returns -1 to signal no combination of components could collectively represent the original class.

To increase clarity, components of Algorithms 4, 5, and 6 were omitted which keep track of which components and widget classes returned the best score (if a score greater than -1 was found). These lists are used to construct or modify the component used by the UI. If no component with score greater than -1 exists, a text component with a warning messaged is used to alert the operator of the deficiency. In practice, if components have been created for all primitive classes, this warning only appears when the recursion limit is small or when an unrecognized class contains itself as a subclass (for instance, a linked list of an unrecognized class).

5.3 Meta-markup

One concern when using situational awareness markup is that an marked up event could occur with high frequency and could result in a decrease in operator situational awareness. This could be the result of the SPN developer and SPN field operator having limited communication during the plan's initial design.

Algorithm 4 Produces a component meeting certain requirements

Input: Enum type, Enum quantity, Class keyClass, Class valueClass
Output: Object component

```
procedure CONSTRUCT COMPONENT SCORE
    score = -1
    switch quantity do
        case SINGLE
            score = GetDirectScore(type, quantity, NULL, valueClass)
            if score == -1 then
                score = GetRecursiveScore(type, quantity, NULL, valueClass, score, 3)
            end if
        case MULTI
            score = GetDirectScore(type, quantity, NULL, valueClass)
            if score == -1 then
                score = GetDirectScore(SINGLE, quantity, NULL, valueClass)
                if score == -1 then
                    score = GetRecursiveScore(SINGLE, quantity, NULL, valueClass, score, 3)
                end if
            end if
        case HASHTABLE
            score = GetDirectScore(type, quantity, keyClass, valueClass)
            if score == -1 then
                keyScore = GetDirectScore(SINGLE, quantity, NULL, keyClass)
                if keyScore == -1 OR valueScore == -1 then
                    keyScore = GetRecursiveScore(SINGLE, quantity, NULL, keyClass, score, 3)
                end if
                valueScore = GetDirectScore(SINGLE, quantity, NULL, valueClass)
                if valueScore == -1 OR valueScore == -1 then
                    valueScore = GetRecursiveScore(SINGLE, quantity, NULL, valueClass, score, 3)
                end if
                score = min(keyScore, valueScore)
            end if
    return score
end procedure
```

Algorithm 5 Direct score

```
1: procedure GET DIRECT SCORE(Enum quantity, Class targetKeyClass, Class target-
   ValueClass, List<Markup> markups)
2:     bestScore = -1
3:     for markupComponent do
4:         score = -1
5:         switch type do
6:             case SELECTION
7:                 switch quantity do
8:                     case SINGLE
9:                         if targetValueClass ∈ SSC then
10:                            score = 0
11:                        else
12:                            for markupWidget do
13:                                if targetValueClass ∈ markupWidget.SSC then
14:                                    score = 0
15:                                end if
16:                            end for
17:                        end if
18:                        if score == 0 then
19:                            for markupOption do
20:                                if markupOption ∈ MO then
21:                                    score++
22:                                else
23:                                    for markupWidget do
24:                                        if markupOption ∈ markupWidget.MO then
25:                                            score++
26:                                        end if
27:                                    end for
28:                                end if
29:                            end for
30:                        end if
31:                    case MULTI
32:                        ...
33:                    case HASHTABLE
34:                        ...
35:             case CREATION
36:                 ...
37:         bestScore = max(bestScore, score)
38:     end for
39:     return bestScore
40: end procedure
```

Algorithm 6 Recursive score

```
procedure GET RECURSIVE SCORE(Enum type, Class targetClass, List<Markup> markups, int recursionLimit)
    if recursionLimit <= 0 then
        return -1
    end if
    subClasses = targetClass.subClasses
    if subClasses.size == 0 then
        return -1
    end if
    minScore = 0
    for subClass do
        score = GetDirectScore(type, SINGLE, NULL, subClass, markups)
        if score == -1 then
            score = GetRecursiveScore(type, SINGLE, NULL, subClass, markups, recursionLimit-1)
        end if
        minScore = min(maxScore, score)
    end for
end procedure
```

In one motivating plan, a SPN gets a list of locations from the operator, then requests temperature measurements be taken at each of those points. The input event corresponding to the completion of a measurement is marked up with "Relevant Area" and "Relevant Information" to include the location in the current view and show the visualization for that data type. In this scenario, the developer of the plan believes the measurements will be spaced far apart and the map manipulations will thus occur infrequently. If the measurements are nearby, then the map will be moved around frequently, increasing cognitive load on the operator.

In a second motivating plan, a SPN has several Display Message output events which are handled by the Message Frame. The developer may not be aware that events are triggered frequently in a particular deployment location, causing many messages to be displayed. Given the high frequency of arriving messages, the Message Frame may be constantly adding and rearranging messages, requiring great effort for the operator to read the messages.

To address these problems in a manner that preserves our ability to be context-aware, we create a new markup which provides guidance to the GUI on how to handle displaying repeated activation/receipt of an output/input event. This abstract markup can then be interpreted by the rules designed by the GUI's designer. Consider scenario 1: a GUI designed for several monitors with a dedicated map for sensor readings and a dedicated map for making decisions may not have a problem when measurements arrive frequently. In scenario 2, the size of the text message display in the GUI will be tightly related to the cognitive load from frequent messages, with larger monitors being able to handle more messages.

We created two new markup items to address these concerns:

Manipulation Frequency: Instructs the GUI that it can suppress manipulation of a component if it has already manipulated it recently.

- Suppress - Discard the event's message if it has already sent a message within a cooldown period determined by the GUI designer
- Timed Batch - Delay adding the event's message until a cooldown period determined by the GUI designer expires

Manipulation Component: Instructs the GUI that it can suppress manipulation of a component if the user has recently used it.

- Focus Suppress: Do not make changes to a component while the operator's focus is on the component
- Timed Suppress: Do not make changes to a component if the operator has manipulated it within a time period

5.3.1 Summary

In this chapter, we presented UIs for an operator in two different domains and an IDE for SPN plan developers. We also presented algorithms which will assist in the reuse of components within and across domains, which could decrease workload for UI developers

and allow for greater impact of human factors research. However, given a sufficiently large library of UI components, it is unlikely that markup alone will be sufficient to distinguish between all the components. It will likely be necessary for the domain and human factors experts to select a subset of components to specify in the domain configuration file which will then be used to construct and manipulate components.

Chapter 6

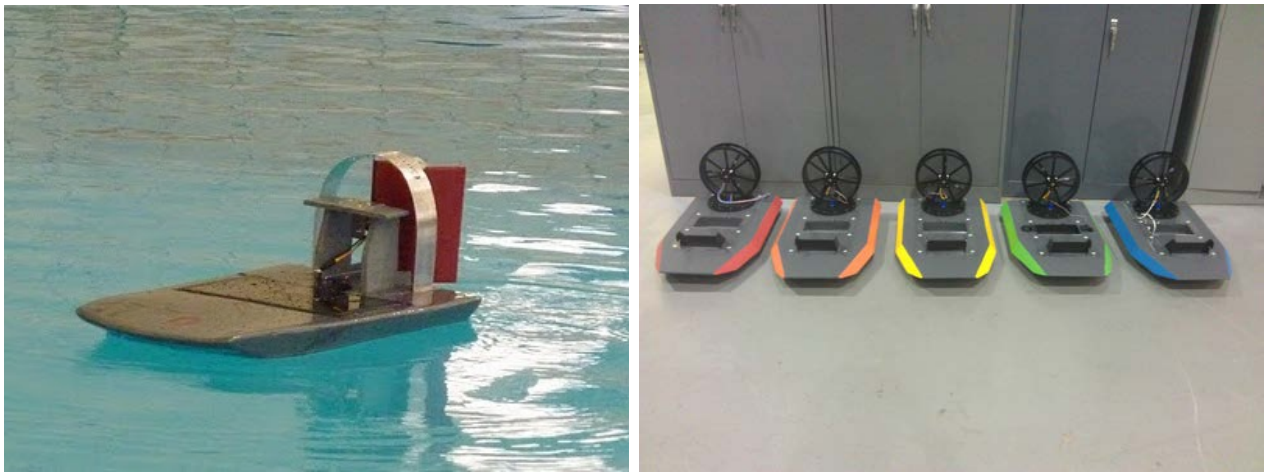
Field Deployment

To test the capability of the SAMI language to specify complex plans, domain libraries were written for the Cooperative Robotic Watercraft (CRW) project [169, 171]. The CRW project investigates the use of small, autonomous watercraft for environmental monitoring and flood response. Relative to other types of vehicles, watercraft are inexpensive, simple, robust, and reliable. In the domain of environmental monitoring, large numbers of inexpensive watercraft can provide high density, routine mapping for a variety of measurements. In response to flooding, flat-bottomed boats using fan propulsion can safely and effectively move through shallow or debris-filled water to provide situational awareness and deliver supplies.

6.1 Boat Development

The first boat, shown in Figure 6.1a, was designed to in 2011 to investigate the use of smartphones in low cost robots. Each boat uses an Android smartphone for communication, either through a wireless local area network or 3G cellular network, GPS location, compass measurements, and a multi-core processor. An application on the Android phone receives high level objectives, such as waypoints, from the user interface or team plan. Control algorithms then generate low level motor and sensor commands to achieve these objectives, and transmit them to an Arduino Mega based electronics board via Bluetooth or USB OTG. The electronics board then interfaces with the motors' electronic speed controllers (ESC), steering servos, and equipped sensors. It supports a wide variety of devices including acoustic doppler current profilers and sensors that measure electroconductivity, temperature, dissolved oxygen, and pH level. The Android application then logs the sensor data with the time and location. A large capacity 8000mAh lithium polymer battery powers the electronics board, motors, and sensors. Figure 6.4a shows the system architecture for the boats.

In 2012, a grant in collaboration with CMU Qatar began, enabling year round development and testing of the boat platforms. Development efforts were focused in Pittsburgh



(a) Fan airboat model (b) First multi-production airboat model

Figure 6.1: Boat propulsion methods

during the spring, summer, and fall. Each winter, a finalized build was mass produced and the constructed boats were shipped to the CMU Qatar campus. A team of 2 to 4 people, including one full time CMU Qatar staff member, would then deploy the boats using the SAMI software to test new functionalities of the vehicle and language. [1]

Changes to the boat designs can be sorted into 4 co-dependent categories. Financial *cost* is a driving factor at odds with the other categories. In general, lower cost components are less reliable and have fewer capabilities. Using the lowest cost solutions will generally result in a larger fleet, but the individual boats will be more prone to failure.

Manufacturability is a driving factor limiting the size of the fleet and usability of the boat as a research platform. Each boat component was obtained in one of 3 ways: as an off-the-shelf (OTS) part, manufactured in-house, or outsourced. OTS availability was generally the most desirable option: by sacrificing customization, the time and financial costs of these components was vastly reduced. OTS parts, in particular smartphones, are a core piece of technology enabling low cost construction of a fleet of autonomous boats. However, many parts are not available in this fashion as there is not a large commercial market for autonomous watercraft. In these cases, they must be custom manufactured either in-house or by an external company. In-house manufacturing is only viable when the tools and skills are available, and then the balance of time versus financial cost must be assessed. For components requiring highly specialized tools and/or skills to manufacture practically, such as custom drivetrain gears, outsourcing is the only option. Committing to a design early reduces the cost through a single bulk purchase, while continually iterating can improve performance while increasing cost.

[1] Other collaborators in boat development included Alex Long, Chris Tomaszewski, and Pras Velagapudi

Capability encompasses hardware and software features of the robot which define the domain tasks it can accomplish. Increasing capability improves the types of team plans that can be run and the general robustness of the individual robots. For example, boat hardware and software was gradually increased to interface with new water sensors. As sensor compatibility was increased, new SPNs could be designed to investigate phenomena related to dissolved oxygen, pH, temperature, and electroconductivity levels and sonar scans. Propulsion ability captures the range of propulsion options, such as fan-driven versus propeller-driven, and the amount of power that can be delivered to the drive system. Low power systems cannot operate in high currents and propeller systems cannot operate in shallow water. Propulsion capabilities affect other capabilities; for example, a boat with limited propulsion will not be able to move a heavy sensor around. Communication strength is the ability for the robots and base station to communicate with each other wirelessly. While equipping each boat with a powerful external antenna will increase their communication range, it will cost more time, money, and reliability to purchase and protect the antenna.

Deployability captures several practical aspects the robot's usability in field deployments. Given a "support team" of 2 to 4 people, one of the primary factors in this category was ease of setup, teardown, and transportation. The reliability of the boats was also important as a robot failure diverts the support team's attention from managing or deploying additional boats. Another factor was durability: the boat hull and propulsion parts inevitably collide with other boats or objects while on the water and being transported off-road. Consumer grade electronics were highly susceptible to permanent damage when in contact with the harsh saltwater, so a high level of protection while on the water and during transport was critical. Repairability of the boat was also important, describing if a sub-component can be replaced without having to replace parent components, and the financial and time cost of obtaining and installing the replacement. It is worth noting that robot development in Qatar at the time of this project lacked many of the resources in Pittsburgh. Online shopping and inventories are rare and there are no companies analogous to McMaster Carr and Sparkfun. When unanticipated items were needed, the quickest solution was often to import the parts from the US, which required two weeks to ship and clear customs.

6.1.1 Winter 2012 Model

The primary focus of boat development for the first winter trip to Qatar was increasing boat deployability while balancing cost.

One of the main improvements to deployability and capability in this model was the custom design of a watertight compartment in the boat hull, as shown in Figure 6.2a. The compartment was sealed using a rubber foam gasket and acrylic "top plate" which was held tight against the hull with bolts. The phone, boat battery, and electronics board were mounted inside the compartment, and watertight rated connectors were used to connect the fan motor and water sensors. This replaced watertight OtterBoxes, which had limited room

(a) Fan driven boat

(b) Differential drive propeller boat

Figure 6.2: Boat propulsion methods

for equipment and fragile tabs.

In a significant improvement to both deployability and manufacturability, the hard-carved foam hulls used in the previous model, shown in Figure 6.1b, were replaced with thermoformed ABS plastic hulls (Figure 6.2a). A bottom and top hull mold were made from hand cut MDF segments which were glued and clamped together. A thermoformer was then used to heat ABS plastic, suction it to the mold, and then "pull" off the molded plastic. The top and bottom plastic hull pieces were then glued together using high strength ABS adhesive and filled with 2 part expanding foam to maintain buoyancy in the event of a hull fracture.

Four of these boats were produced and shipped to Qatar.

6.1.2 Winter 2013 Model

The primary focus of boat development for the 2013 winter trip to Qatar was capability while balancing cost.

While fan-driven boats have many benefits for flat water operation and sensor measurement described previously, several desirable deployment locations in Qatar were in open water with wind speeds regularly between 10 and 30 mph. Given the high winds, low efficiency of fan driven boats, and large cross section by the fan shroud, there were many days where effective experiments were limited or impossible as the boats could not navigate against the current. This motivated the development of a stronger propulsion method as well as research into path planning exploitation of velocity fields. To increase deliverable power and efficiency while reducing wind profile, the fan was replaced with a dual-propeller differential drive, shown in Figure 6.2b. Flex-shaft cables and couples were used to connect each propeller to a high performance, hobby grade brushless DC motor. Flex-shaft cables were selected over solid shafts for their simple integration and superior propeller interface

angle. Significant electronics board development enabled the increased power output and individual control of the ESCs. The motor for the fan propulsion model was air cooled and was mounted sufficiently high above the water to avoid oxidation from the salt water. To cool the each of the DC motors used in the propeller design, aluminum "water jackets" were used, which fit snugly around the motor and are circulated with water to disperse heat. A pump was added into the system, which drew in water from the exterior of the boat to ensure heat could quickly be dumped. An open system using an exterior water source benefits from greater design simplicity and cooling capability compared to a closed cooling system, which recycles a reservoir of water, but is vulnerable to getting clogged from outside particulates and objects. Metal strainers were added to mitigate the buildup of these materials at the water intake.

A new MDF mold was produced featuring a larger electronics compartment to house the drivetrain and a low profile, channeled hull design to improve steerability. The channeled hull design decreases the ability to operate in shallow water as the boat sits lower when at rest, but shallow water operation was not a priority for the propeller driven boat model. Expanding foam was again used to fill the interior of the hull, but a jig was constructed to maintain the gasket interface points during the foam expansion.

To improve deployability and manufacturability, the "top plate" securement was effectively inverted, such that the user tightens down a nut instead of tightening down the bolt. Helicoil was used in the winter 2012 model to add threading for the bolt after the hulls had been joined and filled with foam. Helicoil did not stand up well to the daily wear and tear of repeatedly inserting and removing bolts, with some helicoil failing. When the helicoil failed, the bolt could not be tightened, allowing small amounts of water to flow into the electronics compartment through the bolt hole. Additionally, bolt securement was time-consuming, especially if a hand screwdriver was used instead of a power screwdriver. Given the need to remove the plate to access the electronics, improving this process was important. Wing nuts were used for tightening down the plate as they were found to be the quickest for hand-tightening. Additionally, a neoprene foam was used for the gasket instead of rubber, as it was found to provide a greater tolerance for tightening while maintaining a watertight seal.

While clear acrylic plate was inexpensive and allowed easy viewing of internal components and use of a prism, the heat from the sun caused certain components to overheat in experiments with the 2012 boats Qatar. The addition of thermal blanket material or spray paint to the top plate was tested during the 2012 deployments and both addressed the problem. For ease of application, spray paint was applied to the top plates, except for the prism area.

Another navigation reliability issue encountered was the boat's report yaw orientation reversing occasionally. This was identified as a problem when the boat crested over large waves, which given the orientation of the phone and axis calibration, registered as the boat turning around. An improved phone mount was fabricated to keep the phone at an angle minimizing the effects of waves on the phone's bearing measurement.

Nexus 5 phones were used for these boats instead of Nexus S models for several reasons. The Nexus S phones suffered from a reliability problem where GPS would stop updating during extended deployments. This was identified as a bug in phone's stock Android OS, where commands to prevent the GPS from sleeping did not work. To prevent this problem, software was used to keep the screen on permanently at the cost of battery life. The Nexus 5 supported updated versions of the Android operating system as well as improved GPS and wireless chipsets. To further improve software reliability, improvements to the electronics board were added enabling the use of the USB to-go protocol to communicate with the phone in place of bluetooth communication. The bluetooth stack previously used suffered from reliability issues, with the boat becoming inoperable when a fatal error occurred. This also increased deployability by allowing the electronics board to automate some Android related tasks, such as starting the Android application and removing bluetooth communication related tasks.

A deployability problem encountered in 2012 was the slow disconnecting and reconnecting of cables using Deans electrical connectors. Deans connectors were initially selected to maximize manufacturability, as they are the quickest to solder compared to other connectors which met performance requirements. Deployability was decided to be a higher priority and Deans connectors were replaced with XT-45 connectors. To reduce the decrease in manufacturability, a soldering jig was constructed.

Six of these boats were produced and shipped to Qatar.

6.1.3 Winter 2014 Model

The focus of the development for the winter 2014 model was manufacturability, as nearly two dozen would be built, and deployability, as the fleet would need to be deployable by 2 to 4 people.

To reduce cost, the Samsung Galaxy S3 mini was selected as the phone model for the 2014 model.

Despite the use of a soldering jig, XT-45 connectors were still slow to wire and were slow to connect and disconnect compared to other connector types. To improve manufacturability and improve deployability, high power electrical connectors were replaced with Anderson Powerpole connectors. Figure 6.4b shows the electronics board connected to its top plate, with Powerpole connectors inside the compartment and IP67 rated watertight connectors outside the compartment. Powerpole connectors require the use a special crimping tool which removes the need to solder wires to the connector.

To manufacture dozens of hulls, a higher quality mold was required which would sustain repeated thermal stress from thermoforming. The new mold, show in Figure 6.3b, was made from vertical layers of high density wood cut using a CNC router and then coated with epoxy. Finally, a release wax was added to regions of the mold with concave features to relieve stress when separating the molded plastic from the mold, reducing probabilities

of the plastic cracking. The new hull was designed to have greater buoyancy to support large sensor payloads. In addition, a thicker ABS was used to increase durability during transportation and allow safe operation in near-freezing water.

To improve manufacturability, boats were filled with blocks of styrofoam instead of air bags. Styrofoam blocks were more cost effective and could be easily taped down, increasing the amount of fill and reducing the time required to align the top and bottom hull pieces.

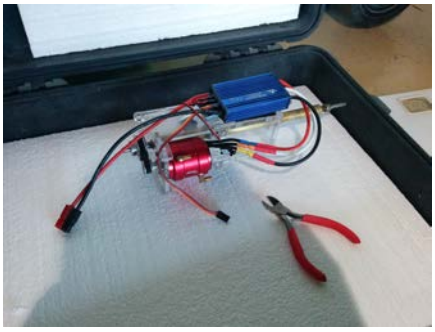
One of the largest improvements to fleet deployability was the development of rotating "tabs" connected to the hull which replaced the bolt and wingnut assemblies in holding lids against the hull. The tabs (Figure 6.3c), combined with a new neoprene foam gasket material, allowed a lid to quickly be locked in place by depressing the lid against the gasket and then rotating each of the tabs into a position above the lid.

In the 2013 model, phone mounts required some careful maneuvering to slide the phone into position. To increase deployability by speeding up phone mounting, a velcro mount was used where velcro on the back of the phone attached to velcro at the front of the compartment. This greatly improved deployability, but reduced capability as the phone's camera feed was no longer useful.

During field deployments, particles in the water flowing through the cooling system would occasionally get stuck at tube junctions. If enough particles built up in a junction, it would cause the tube to separate from the junction, resulting in saltwater flooding the internal compartment. This was not detectable until the boat stopped working because the entire compartment was flooded, corroding and destroying all the electronics. To increase reliability, a new cooling system was designed which removed the use of external water. This cooling system used pre-primed distilled water to draw heat from the motors and ESCs and an aluminum heat pipe on the underside of the hull to exchange heat with the external body of water. The likelihood of a pipe disconnecting was greatly reduced as no foreign particles could enter the system, and if a pipe did disconnect, the small amount of distilled water would not permanently damage the electronics.

A second source of saltwater entering the internal compartment and destroying electronics was imperfect sealing in the drive system, as water could enter through the flex-cable tube as grease was dislodged. To increase reliability and add more storage space for additional batteries, a second compartment was added to the hull design in front of the original compartment to isolate any water leakage to the rear compartment. The rear compartment housed the motors and ESCs, and the front compartment housed the more water-sensitive boat battery, phone, and electronics board. An elevated channel was used to run wires between the two compartments.

The 2013 drive system suffered from many deployability problems. The set screws required to couple the motor shaft to the flex cable required constant tightening. In addition, the hobby grade flex cables used were not designed for use in extremely salty water and would rust if not greased and cleaned daily. Overtime, exposure to the elements and retightening would degrade the cable until it required replacement. This required disassembling and

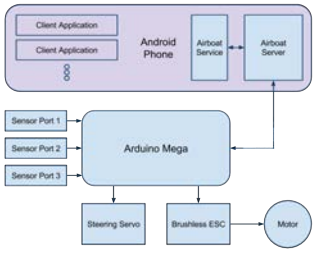


(a) Powerpod

(b) Bottom hull mold

(c) Plate tabs

Figure 6.3: 2014 boat components



(a) System architecture of a boat

(b) Electronics board

(c) Boat fiducials

Figure 6.4: 2014 boat components

reassembling many components and was not scalable to a larger number of robots without increasing the size of the support team. Furthermore, while the boats performed well in high current, when larger payloads were added, such as side scanning sonar systems, the boat had difficulty moving against the current. To address these problems, an new drive system was designed using a solid shaft instead of a flex cable. While this led to a less efficient propeller interface with the water, a solid shaft increases energy efficiency and allows the boats to be easily stacked for transportation. The shaft was greased and sealed with a brass tube, protecting the shaft from the elements and eliminating regreasing tasks. Custom gears to lower the gear ratio were outsourced to improve operation in high current and with large payloads and eliminate shaft couplers. Each drive system was housed in an acrylic enclosure and the entire"power pod" (Figure 6.3a) bolted into the posterior hull compartment, greatly simplifying the replacement process in the event of failure.

Twenty-three of these boats were produced and shipped to Qatar.

(a) Doha Corniche (b) Katara Beach (c) Fuwayrit Beach

Figure 6.5: Deployment locations

6.2 Test Sites

In this section we describe the locations the boats were deployed in and the unique challenges each offered. In each of these environments, a high power wireless antenna was used to create a wireless network connected the base station laptop and boat phones. The operator uses a SAMI compatible GUI described in Chapter 5 to instantiate SPN plans, monitor their execution, and provide input as necessary.

Doha Corniche

Figure 6.5a shows the initial test site on the Doha Corniche. Because the Corniche is near the Emir's palace, access for boat experiments required a great deal of paperwork and had rigid hours of access. Quadrotors were prohibited from operating on the premises, which reduced our ability to film experiments. The water along the Corniche often experienced high winds which proved to be too much for the fan-propelled boats to drive against, even with moderate amounts of switchbacking. Due to the complicated access, limited parking, and traffic congestion in the area, new deployment locations were investigated.

Katara Beach

Figure 6.5b shows the primary test site for propeller driven boats, Katara Beach. Katara Beach is a large beach site and has many areas with limited access for boat trailers to load and unload boats, jetskis, and even rowboats. While swimming is limited to a small area, boat traffic is common, requiring operator vigilance to detect and react to boats approaching the deployment location. Buoys, anchoring rope, and construction platforms are moved on a day to day basis, presenting new obstacles both for the boats and the operator's perception. Removing a boat from the water at Katara beach is complicated by boat wake, high winds, and a concrete ramp extending into the water, which can damage hulls and propellers if the fall of the tide has been underestimated. However, in the event of catastrophic hardware failure, lifeguards on jetskis are happy to help recover the boat.

(a) Transportation of equipment (b) System setup at Katara Beach

Figure 6.6: Deployment setup

Fuwayrit Beach

Figure 6.5c shows the secondary test site, Fuwayrit Beach. Fuwayrit Beach is a relatively remote beach on the east coast of Qatar which lacks the boat traffic found at Katara Beach. The geographic properties of parts of Fuwayrit Beach shelter it from the wind found on the Corniche and Katara Beach, making it a good location for windy days or when private events limit access to Katara Beach. In particular, Fuwayrit Beach has a large sand bank with shallow water where navigation is largely unaffected by the wind and makes deployment and recovery simple. However, the falling tide quickly exposes it, forcing manually recovery of any boats left in the sand and redeployment to a more exposed part of the beach.

6.3 Deployed Plans

In this section, we will discuss a set of plans used during deployments in Qatar. These plans use a subset of the output and input events and markup described in Appendix .2. Figure 6.7 shows a timeline of plan usage during a 3 hour deployment at Katara Beach. We will describe and analyze 4 of these plans: Connect Boat, Connect and Station Keep, Explore Area, and Monitor Connectivity.

6.3.1 Connect Boat

The "Connect Boat" plan shown in Figure 6.8 was used to connect the operator GUI to a boat's Android phone server. To reduce the amount of time required by the operator, the SPN variable system was used to store global variable definitions of the "BoatProxyId"

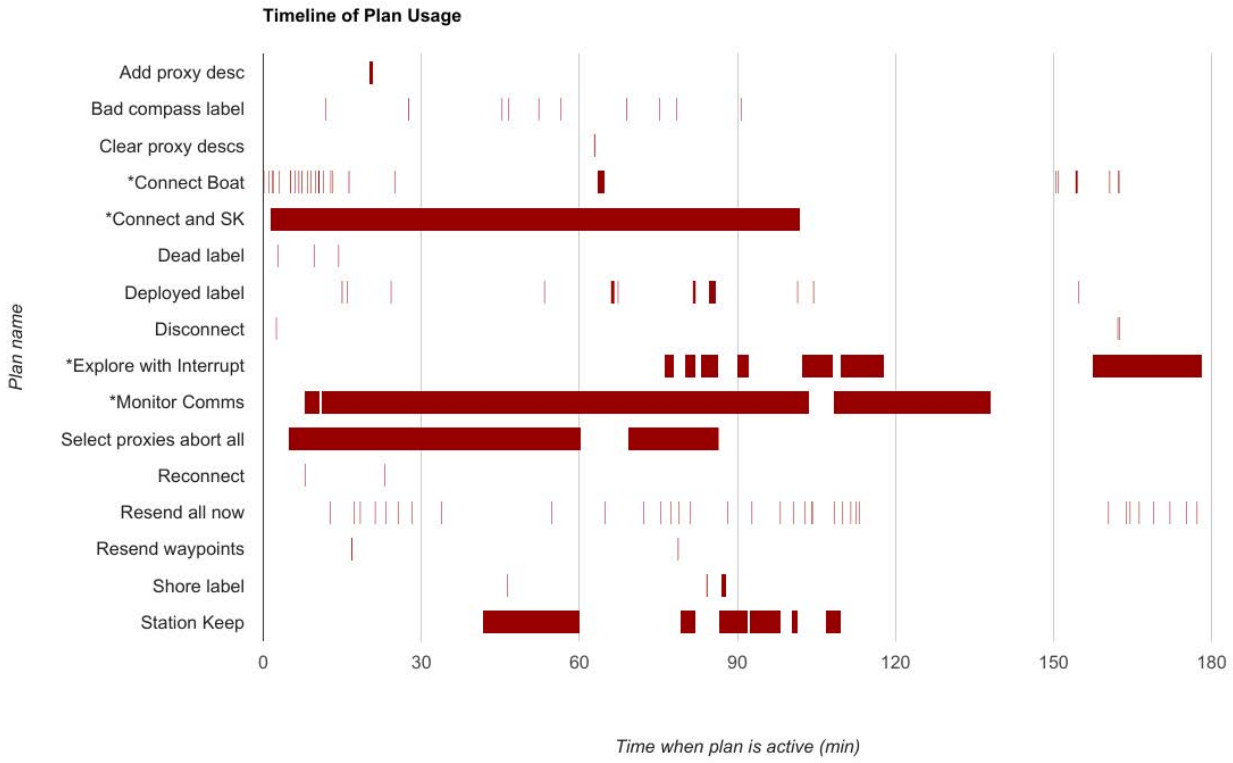


Figure 6.7: Plan invocations during a 180 minute deployment at Katara Beach

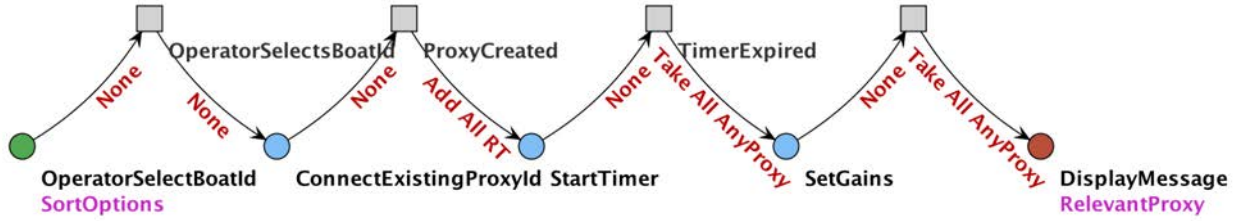


Figure 6.8: Connect Boat SPN

# Run	Total Duration	Avg Duration
134	18min 56s	8s

Table 6.1: Runtime statistics for Connect Boat over 2 days

class, which consisted of the boat's static IP address, name, and visualization color. Instead of manually entering this information, instead the operator was presented with a list of all BoatProxyId definitions in scope of the plan and selected the desired boat as shown in Figure 6.9. After the operator selects the boat ID, the SPN connects the GUI to the server, waits for a connection to be established, sets steering and throttle gains according to a global variable, then notifies the operator the boat is connected. The use of a SPN variable for the gains allows for variants of this plan to be easily created to account for different drive systems and daily variations in wind and tide forces, e.g. "Connect Fan Boat on Windy Day". Rather than have each variation of this plan in a single SPN library, instead a SPN library for each deployment scenario is be created, e.g. "Katara Beach, Windy."

To expedite boat ID selection by predictably presenting IDs, markup is used to specify that IDs should be sorted alphabetically. Table 6.1 shows statistics collected over 2 days demonstrating how often boat servers were connected to and the average time between spawning the plan instance and the plan instance ending. Boats could be connected to multiple times in one day due to circumstances such as the GUI being restarted or the operator disconnecting from a boat server. An additional variant of this plan used the "ProxyAddDescription" output event to set the boat's description to *SHORE* to note that the boat is currently on the shore. This description can be used by the GUI to filter data and options in other plans.

6.3.2 Connect and Station Keep

The "Connect and Station Keep" plan shown in Figure 6.10 was used to incrementally put boats into the water. It was designed to require only short bursts of operator attention so that the operator could assist in preparing other boats for deployment. When the plan is started, the operator selects a location in the water which connected boats will be instructed

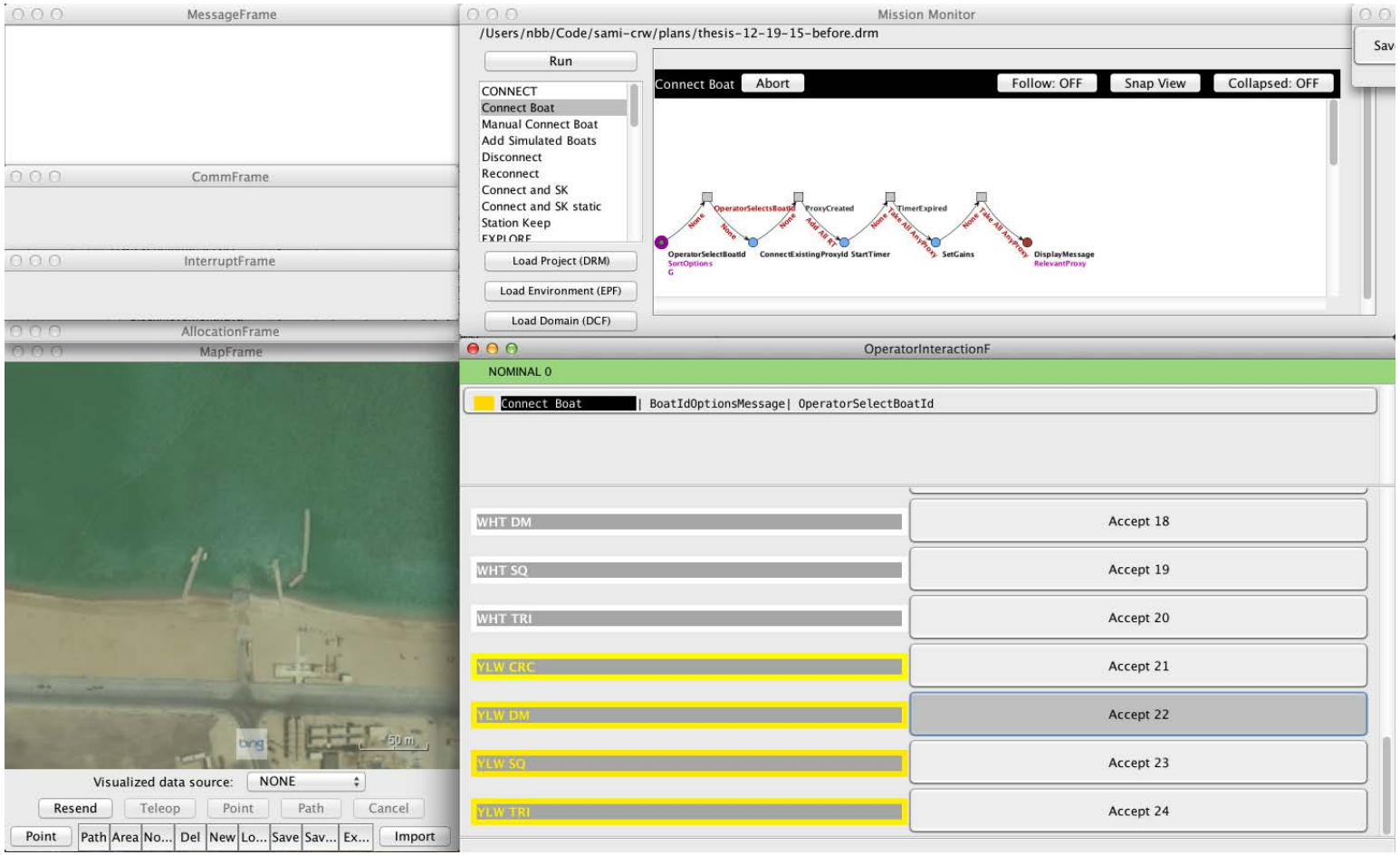


Figure 6.9: Alphabetized boat ID list interpreted during Connect Boat SPN

to station keep near. The operator is then instructed to select a boat they wish to deploy into the water from a list of boat servers the GUI has connected to. For the boat of interest, the operator or team member performs a brief, visual diagnostics check on the shore, places it in the water, then selects it from the list in the GUI. The SPN then calculates a unique location near the specified location for that specific boat to station keep on to minimize collisions with other boats. The proxy's token is then moved into a place which spawns a Station Keep sub-mission, which references this unique location. For various reasons, such as dynamic hazards and changing tides, it is valuable to be able to change the central station keeping location without starting a new plan and re-adding each boat. This is achieved through an operator interrupt which moves a token a the place with a "RedefineVariablesRequest" output event. The SPN has the central station keeping location's variable name specified as the variable name to redefine. At run-time, the variable's value is retrieved, and an appropriate component is constructed based on the value's class so that the operator can define a new value. Once a new value has been received and saved to the variable name, a "RedefinedVariablesReceived" input event is generated. At this point, lookup table of unique proxy locations is regenerated, using the updated central station keeping location value and a copy of each proxy token that has been selected so far by the operator. The next time a boat's Station Keep sub-mission performs a distance comparison, it will compare it to the newly computed location.

To speed up boat selection, two types of markup can be added to the OperatorSelectBoat output event. SortOptions markup is used to sort the list of boats chronologically. The GUI interprets this markup by sorting the list of proxy tokens used to activate the event chronologically by the time the proxy was first created (i.e., when the GUI first connected to the boat), with the most recently connected boat at the top of the list. This markup is very useful in the field, where a boat is typically connected via a "Connect Boat" plan and then immediately selected in the "Connect and Station Keep" plan, where it will be at the top of the list. Another useful addition is using the FilterOptions markup to omit options not relevant to *SHORE*. When combined with other plans setting proxy descriptions, this can prevent the operator from sifting through the list of boats which are in the water ProxyAddDescription updates the proxy description as *WATER*. As adding additional boats is less important than resolving existing problems, the "Add another?" OperatorApprove output event message is marked up with low priority.

A video of this plan in use at Katara Beach with an overlay of the operator GUI and a simplified SPN state representation is viewable at `https://www.youtube.com/watch?v=l5Qhp1JSoNI`. Figure 6.11 shows a selection of frames from the video[2]. To improve readability, events and edge requirements are hidden in the SPN state representation, which is instead summarized using vertex labels and tokens. The video demonstrates the importance of the station keeping formation and redefinition interrupt as well as the ease of adding boats

[2]This video was accepted to IJCAI-VC-2015.

# Run	# Add Boat	Avg Time Between Add	# Redefine Location	Total Duration	Avg Duration
8	208	110s	30	3hr 34min	26min 45s

Table 6.2: Runtime statistics for Connect and Station Keep SPN over 2 days

and low level of required operator involvement.

Figure 6.12 shows a time window of GPS paths for boats executing the Connect and Station Keep in a eastward flowing current. A time window was selected to improve visual clarity, hence only a subset of boat paths include movement from the shore to the station keep location. Note that the dock ramps present in the satellite image were not present at this deployment and the tidal boundary was farther out. This visualization displays data from several invocations of Connect and Station Keep which used slightly different station keeping points. Each invocation uses a Proxy Compare Distance threshold of 10m - this distance is shown at scale in bottom right corner of the figure.

Table 6.2 presents statistics for the plan collected over 2 days. These statistics show that the plan was active during a large portion of the deployment (3 hours and 34 minutes over the course of the 2 days) and the operator interrupt used to change the station keeping location was used frequently (a total of 30 times over 8 instances of the SPN). On average, boats on the shore were added at the rate of 1 boat per 1 minute 50 seconds, allowing enough time to perform a quick diagnostics check and place it in the water. This diagnostic check consisted of the following actions:

- GPS check: Check that boat's reported location on the GUI is reasonable
- Compass check: Check that boat's reported heading on the GUI is reasonable while held pointing in each cardinal direction
- Motor check: Check that propellers generate air movement in the correct direction and magnitude when teleoperated to execute a left turn and right turn

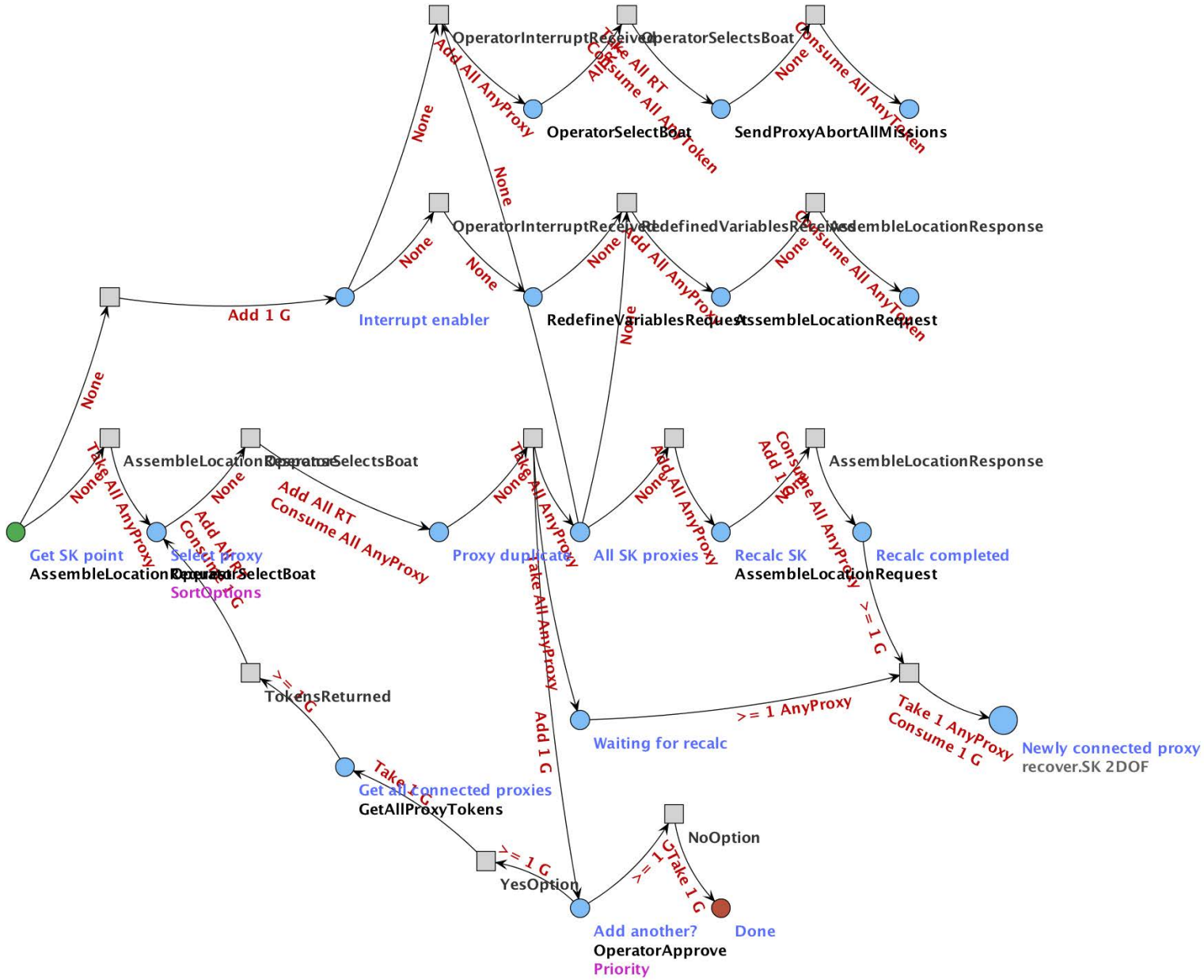
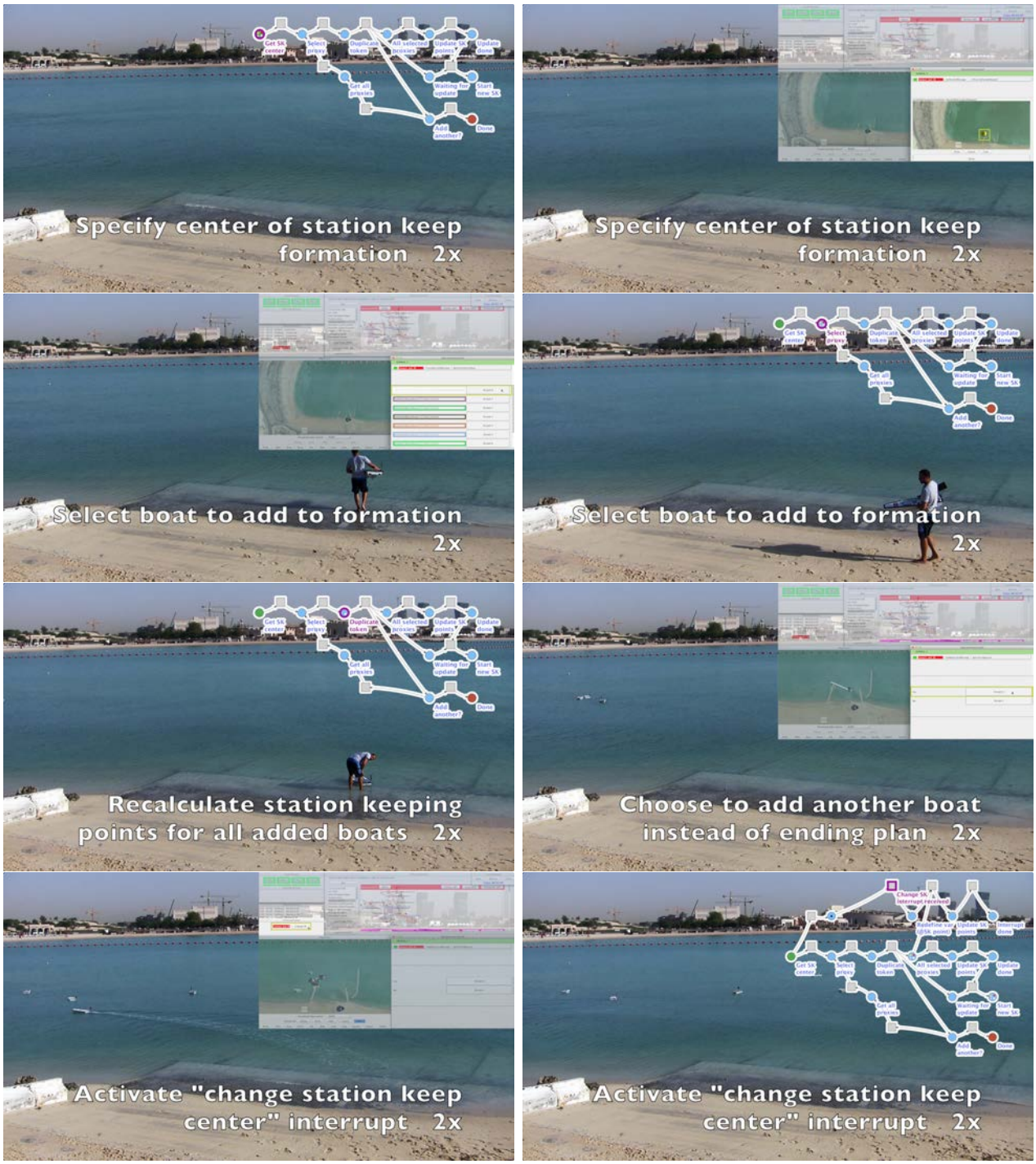


Figure 6.10: Connect and Station Keep SPN


Specify center of station keep formation 2x
Specify center of station keep formation 2x
Select boat to add to formation 2x
Select boat to add to formation 2x
Recalculate station keeping points for all added boats 2x
Choose to add another boat instead of ending plan 2x
Activate "change station keep center" interrupt 2x
Activate "change station keep center" interrupt 2x

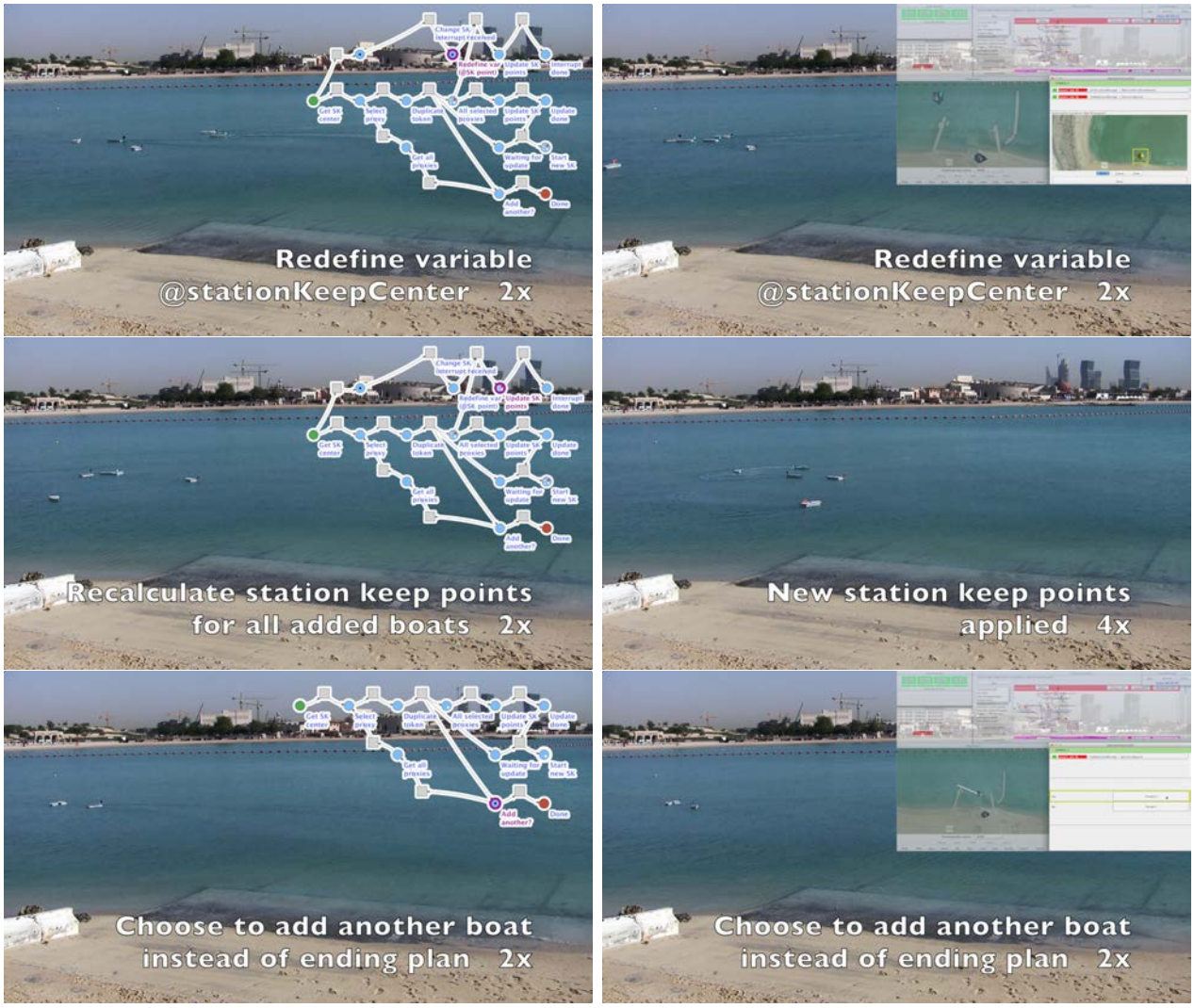


Figure 6.11: Video frames from Connect and Station Keep video

Figure 6.12: Boat paths during connect and station keeping

6.3.3 Explore Area

The Explore Area plan, shown in Figure 6.13, was used to get an operator-defined area of water and divide it among an operator-selected set of robots. Upon starting the plan, the operator is asked to define the area to be explored. After defining the area, the operator selects a subset of the robot team to be used for the exploration. A subsection of the area and "lawnmower" path through it is assigned to each of these robots. When all of the robots have finished their path, the plan notifies the operator and ends.

The Explore Area plan contains several operator interrupts. The "Team avoid boat" interrupt was used to address the situation where another watercraft needed to pass through the area being mapped. Prior to adding this interrupt, the procedure in this situation was to cancel the plan, start a new plan to move the boats out of the way, and then restart the plan. This solution required a great deal of operator attention and resulted in remapping the previously explored area unless the operator meticulously defined a new area carving out each robot's partially completed lawnmower pattern. Instead, when the speedboat interrupt is activated, a copy of each proxy token which was selected to explore the area is used to activate the "speedboat.SK 2DOF" sub-mission. In this sub-mission, shown in Figure 6.14, the operator selects a location clear of the approaching boat's path. Similar to the Connect and Station Keep plan, it then selects a unique, nearby location for each boat in the plan and conducts station keeping centered on that location. This plan retains the ability to

use an operator interrupt to redefine the central location if the boat changes course or there are multiple incoming boats. This plan has 3 notable differences compared to the station keeping in Connect and Station Keep. First, when robot's are beyond the distance threshold from their unique location, the "ProxyGotoPointAndBlock" output is used in place of "ProxyGotoPoint." ProxyGotoPointAndBlock behaves similar to ProxyGotoPoint except that, after arriving at the destination, any other waypoints in the robot's waypoint queue originating from other SPNs are "blocked" from executing. When the SPN containing the activated ProxyGotoPointAndBlock terminates, this block is removed. This behavior is necessary to prevent the robot from draining its battery by entering a loop where it travels to its "avoid boat" station keeping point, then drives back towards its assigned exploration area until it reaches the "avoid boat" distance threshold, where it then travels back to its station keeping point. A second difference is the use of Priority markup on ProxyGotoPointAndBlock, which is set to CRITICAL. Its interpretation results in the robot immediately executing the command instead of adding it to the end of the robot's queue of commands.

The third difference is the termination condition. In Connect and Station Keep, the station keeping sub-missions have no termination condition and are instead terminated when the parent plan ends. As detailed in Chapter 3, when a mission ends, all of its sub-missions are also ended. The mission will end either when the operator answers "No" when asked if they want to add another boat or when the operator chooses to abort the mission. In the Avoid Boat sub-mission, an "OperatorApprove" output event is used to ask the operator if it is safe to terminate the mission. If "Yes" is selected, the mission is ended. If "No" is selected, the question is asked again. Note that this method of ending the plan was design with knowledge of how the runtime GUI (presented in Chapter 5) would handle interactions. For a GUI which uses a "queue" of interactions and a dedicated space where the operator can browse through the queue, the operator can "ignore" interactions without compromising the rest of the GUI's functionality. If the GUI instead used "pop-up message" style of interaction, this method of terminating the plan would be unusable as the message would appear indefinitely until "Yes" is selected. Adding a system timer event before asking the question again would decrease the severity of this problem, but the operator would lose the ability to end the plan on demand. A better solution which would work for both styles of GUIs would be to instead use an operator interrupt to end the plan.

OperatorSelectBoat has 3 pieces of Markup: FilterOptions, SortOptions, and Default-Selection. The Filter Options markup is set to only *INCLUDE* options associated with *DEPLOYED*. For OperatorSelectBoat, the GUI interprets this markup by looking at the name of the boat and the description set using the "ProxyAddDescription" output event. If a boat has been selected in the Connect and Station Keep plan, its description will be set to DEPLOYED unless it has encountered an error, which will be discussed below in the "Monitor Communications" plan. The Sort Options markup is set to *ALPHABETICAL* to present boat names in a predictable manner. The Default Selection markup is set to *ALL*

# Run	# Avoid Interrupt	Total Duration	Avg Duration
13	6	57min 12s	4min 24s

Table 6.3: Runtime statistics for Explore Area SPN over 2 days

as typically the entire team is used. The result of these three markups is that only the boats currently in the water and not experiencing errors are selectable, and these boats are presented alphabetically and by default their boxes are "checked."

The DisplayMessage output events each have a "RelevantProxy" markup set to *ACTIVATING TOKENS*. When tokens enter the place and activate Display Message, a message is is placed in the operator GUI's Message Frame as described in Chapter 5. When this message is clicked, the markup is interpreted by zooming out the Map Frame if necessary to ensure the markers for all the boats whose proxy tokens activated Display Message are visible and also highlights those markers.

Figure 6.15 shows images of the GUI and a team of boats executing the Explore Area plan. In Figure 6.15a, the area has already been defined and the operator is asked to select the boats to be used. The yellow area in the map does not correspond to the defined area, it is a separately defined area corresponding to obstacle free space where high WiFi connectivity was observed. As noted, the markup for "Operator Select Boat List" is interpreted such that only "deployed" boats without known problems are displayed, and by default all those boats are selected. In Figure 6.15e, the operator is asked to provide definitions for the Avoid Boat interrupt: the location for station keeping and the timer duration for checking a boat's distance from the location. Two locations (shaded yellow spheres) are seen in the map as the expert operator clicked once in the map, then decided to choose a location farther north and clicked a second time, knowing that the GUI would return the last provided location. Alternatively, the operator could use the "Clear" button to remove all currently clicked locations if they wish to change their initial selection.

Figure 6.16 shows a time window of GPS paths for 4 boats executing the Explore with Interrupt plan. In this particular instance, the white robot encounters a compass failure and begins driving in circles. The operator invokes the Proxy Abort interrupt for the boat and then teleoperates it to the shore. The yellow boat completes its locally stored set of Explore Area waypoints, but later encounters a communication problem and is unresponsive to operator commands. The current carries it to the western shore, where it is recovered and walked back to the operator station.

Table 6.3 shows runtime statistics for the Explore Area plan. Over 13 invocations of the plan, the Avoid Boat interrupt is used 6 times. One invocation is substantially longer than the others due to a large area being selected. Some invocations are shorter as they are used for reasons other than mapping such as testing motor resistances and PID values and increasing network load.

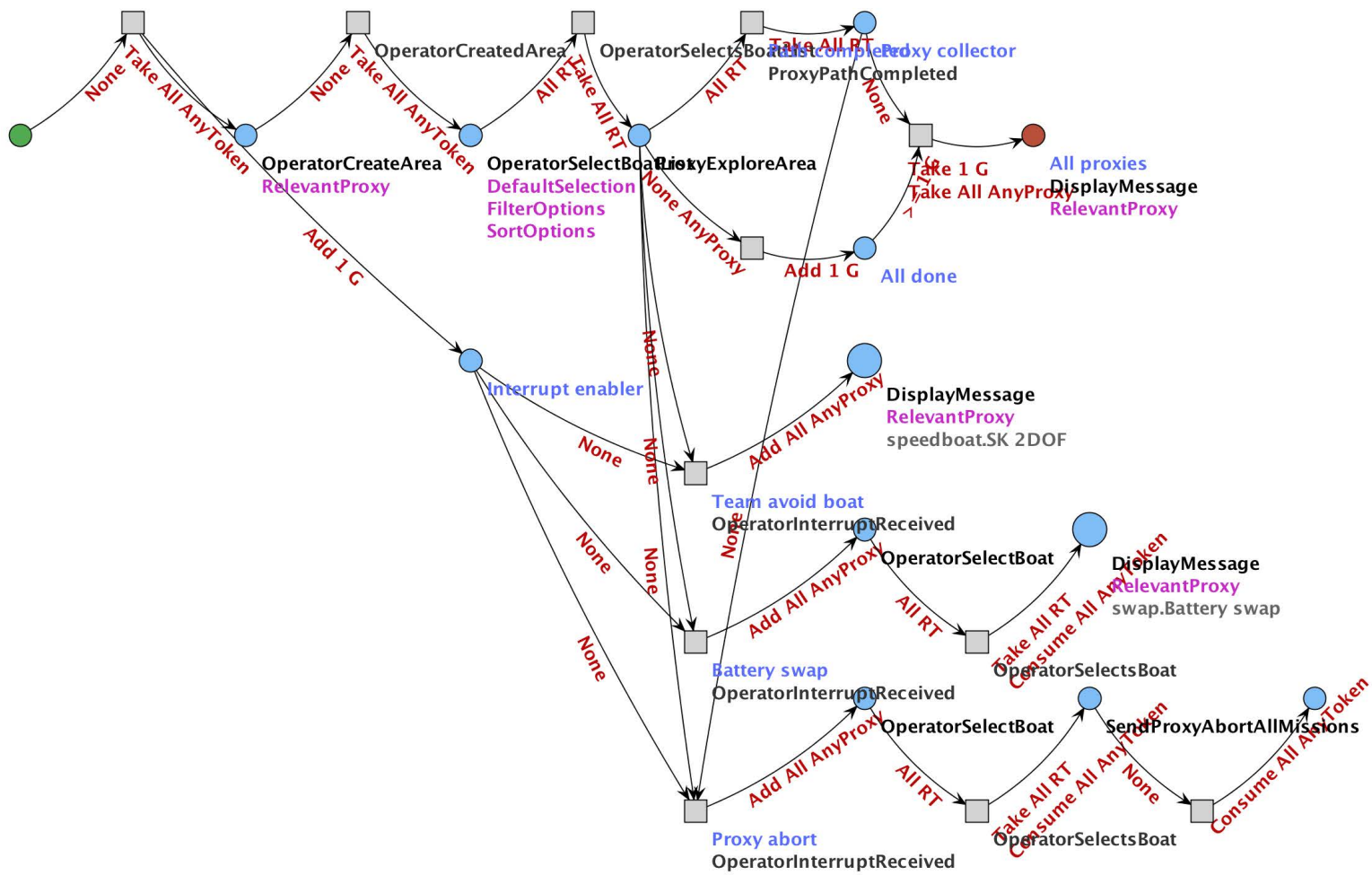


Figure 6.13: Explore Area SPN

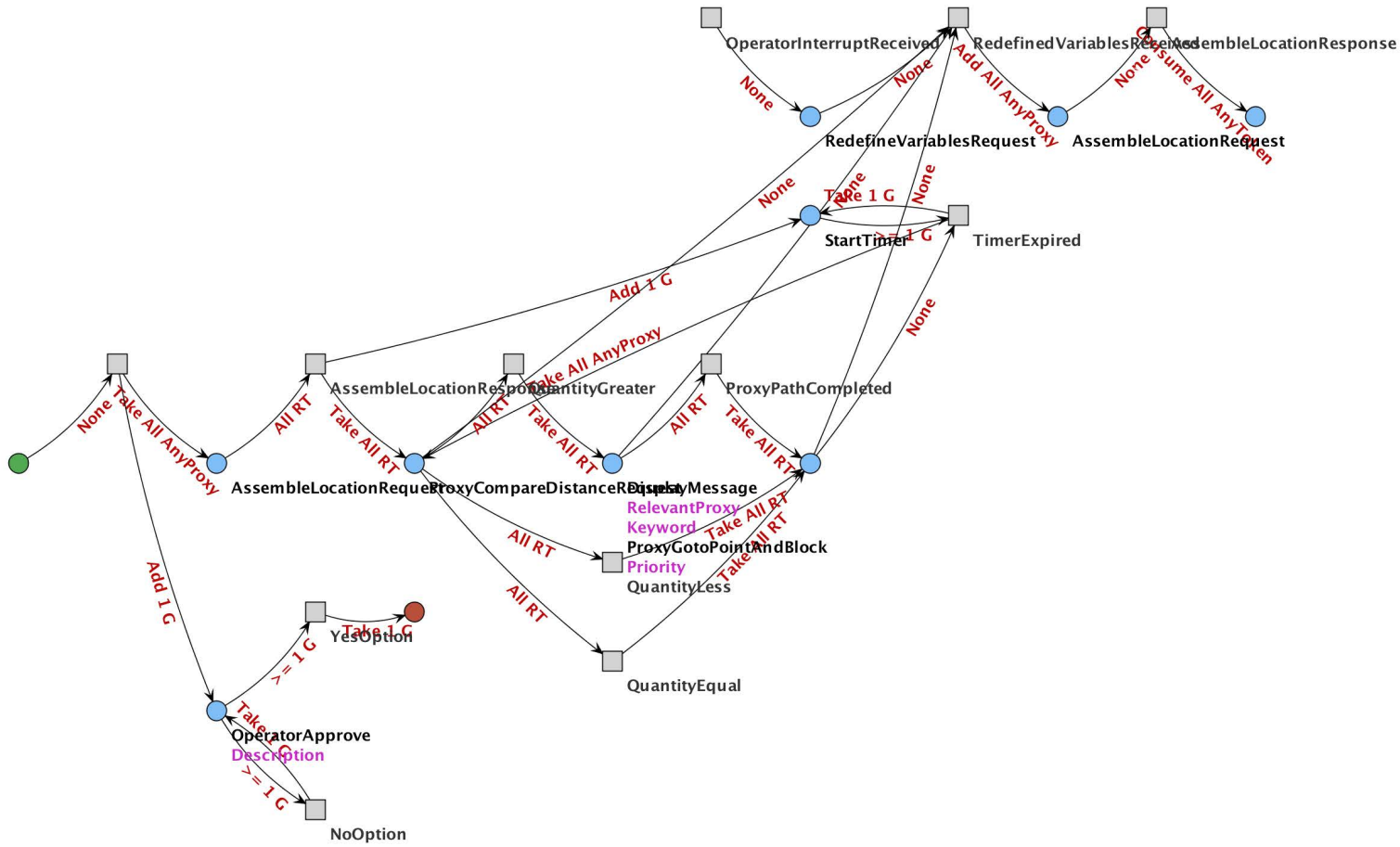
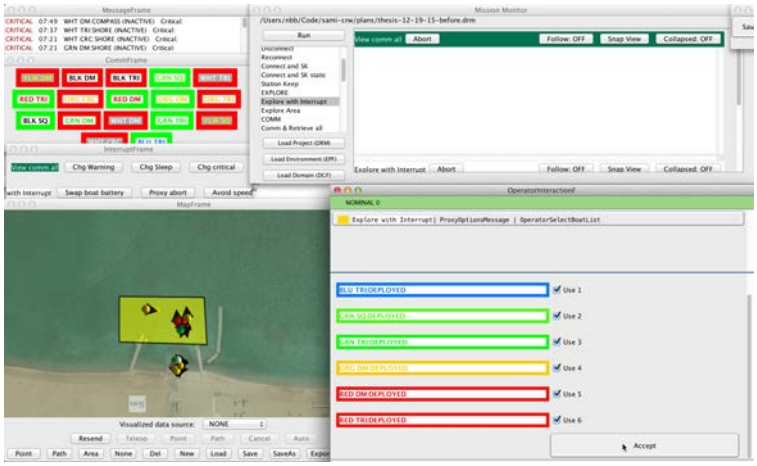


Figure 6.14: Speedboat SPN used as sub-mission in Explore Area

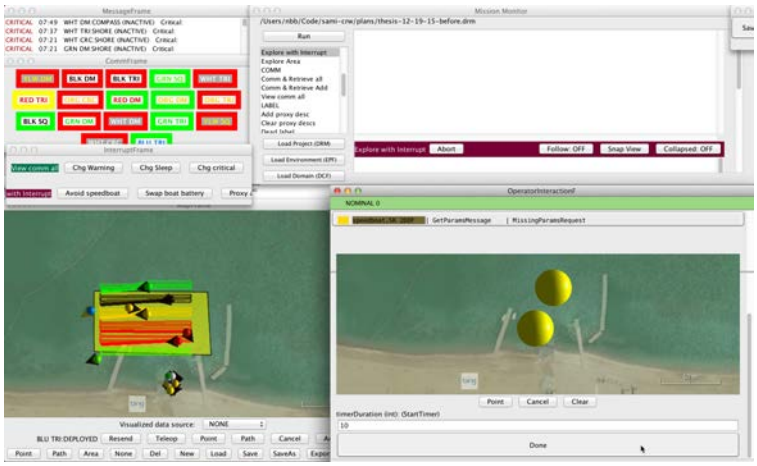


(a) Selecting boats to use for exploration

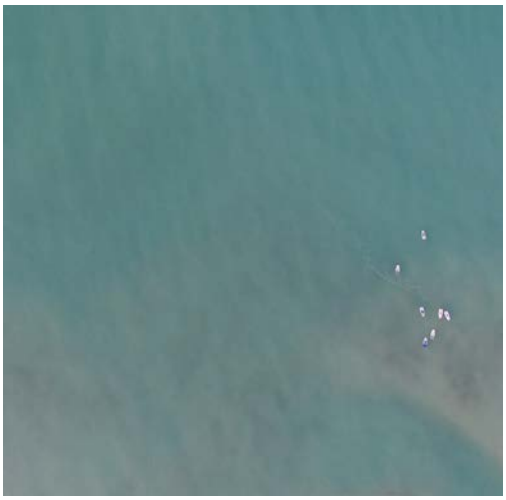
(b) Selected boats moving into position

(c) Monitoring exploration execution

(d) Executing "lawnmower" pattern



(e) Invoking "Avoid Boat" contingency



(f) Boats at safe location

Figure 6.15: Explore Area SPN executing

Figure 6.16: Boat paths during lawnmower exploration

6.3.4 Communications Monitoring

The "Communications Test" plan shown in Figure 6.17 was designed to visualize and manage wireless connectivity problems. Wireless performance during field deployments was poor compared to laboratory and shore side tests. It was hypothesized this is largely due to the electromagnetic absorption properties of water and mounting height of the smartphone. Before this plan was designed, an experiment was performed to compare the maximum range from the omnidirectional base station antenna where a connection to a single boat could be maintained. A single boat was placed in the water and slowly driven away from the shore until connectivity failed. The boat was then brought back to shore, reconnected, and then carried at waist level along the shoreline until connectivity failed again. Figure 6.20 shows the results of this experiment, where the range in the water (90m) was less than half of that on the shore (200m).

In deployments with team sizes greater than 6, frequent and dramatic connectivity problems were encountered within the measured range. Waypoint commands sent to nearby boats were often not received and pose updates were received by the GUI less frequently. It was hypothesized that the water absorption also impacted the bandwidth of the router by increasing packet failure. This could be compounded by increased packet collisions due to tight grouping of increasingly large formations. If the operator did not realize boats were

not received commands, the boats would drift farther from communication range until communication was impossible. Recovering the boat then required waiting until it drifted to another shore, or using motorized watercraft to retrieve it.

This plan was designed with two purposes in mind. First, the plan should provide the operator with a quick way to check connectivity across the entire team to recognize patterns or conditions which affect communication quality. This plan was designed for quick experimentation without having to constantly retrieve and deploy boats to update firmware and copy data files. Second, the plan should prevent boats from drifting away if the operator was is busy with other tasks or away from the computer.

In the plan, robot tokens are moved between places corresponding to increasing levels of connectivity problem severity. When data is received from the robot, in the form of its pose, the robot's token is moved back to the place representing nominal connectivity. Operator interrupts are present so the timers controlling movement between the places can be adjusted on the fly. If a robot remains in the "Critical" communication place for a period of time, the robot will be told to move to a recovery location defined when the plan is selected. This waypoint command is sent continuously until the robot arrives at the recovery location as the poor communication level greatly increases the probability that a command sent only once would not be received.

Operator select boat list uses Sort Options markup to alphabetize the boat list for ease of selection. Each Display Message event uses 3 types of markup. Proxy Status is used to note the status of the proxy in the Communications Monitoring plan, and its definition of Nominal, Warning, or Severe corresponds to the Nominal, Warning, or Critical place the Display Message event is on. The GUI interprets the Proxy Status markup by changing the color of the robot's border in the Comm Frame: red if its status is Critical in any plan, yellow if its status is not Critical in any plan but is Warning in at least one, and green if its status is not Critical nor Warning in any plan. Relevant Proxy is interpreted by the GUI when the message is clicked in the Message Frame. When clicked, the Map Frame is zoomed out (if necessary) so that the robot corresponding to the message is visible and the robot is also highlighted. The Display Message events corresponding to Warning and Critical communication states also have Priority markup set to High and Critical, respectively. This ensures that newly arriving messages about nominal connectivity for one boat do not displace slightly older messages about critical connectivity to another boat. The Keyword markup requires a String "keyword" value indicating that events with the same keyword markup should be grouped together. The Message Frame interprets Keyword markup by maintaining a table mapping each encountered keyword to the most recent message marked up with that keyword. When a message with a keyword is received, it replaces the previous message for that keyword (if one exists) in the table. This results in the old message being removed from the Message Frame and the new message being added. If Relevant Proxy or Relevant Information markup is also present on the event, the table lookup contains those definitions (the proxy ID and information type, respectively) in addition to the keyword. This markup serves two purposes

for Display Message. First, it prevents the high frequency connectivity messages from being displayed separately, which would quickly obscure other messages. Second, through use with Relevant Proxy markup, it prevents old, high priority messages about critical connectivity to a boat from being placed higher than newer messages about nominal connectivity being established with that same boat. The ProxyGotoPoint event in the "Go home" place also uses Priority markup. As in the Avoid speedboat interrupt in the Explore Area SPN, this is set to CRITICAL to indicate the waypoint should be executed immediately.

Figure 6.18 shows the GUI executing the Monitor Comms SPN. The router was turned off to simulate a total loss of communication, resulting in each boat's border in the Comm Frame to turn red as seen in Figure 6.18a. In Figure 6.18b, the router has been turned on and boats begin reconnecting and restoring nominal connectivity.

Figure 6.19 shows boat paths during a time window where the base station router stopped functioning. The operator recognized the failure and was able to set up a backup router, and the "Resend WP" events were able to regain control of all but 5 boats before they drifted out of range of the backup router.

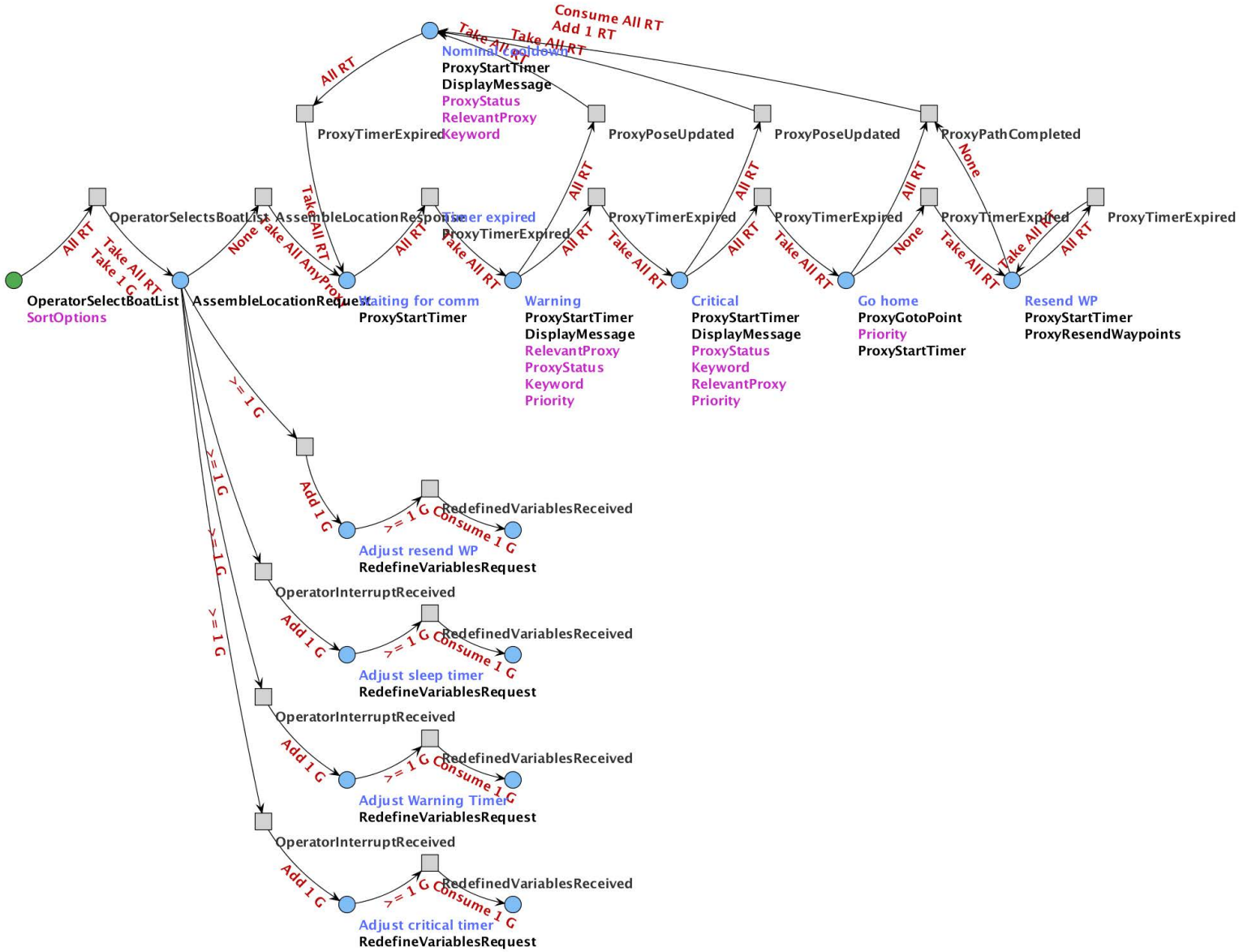
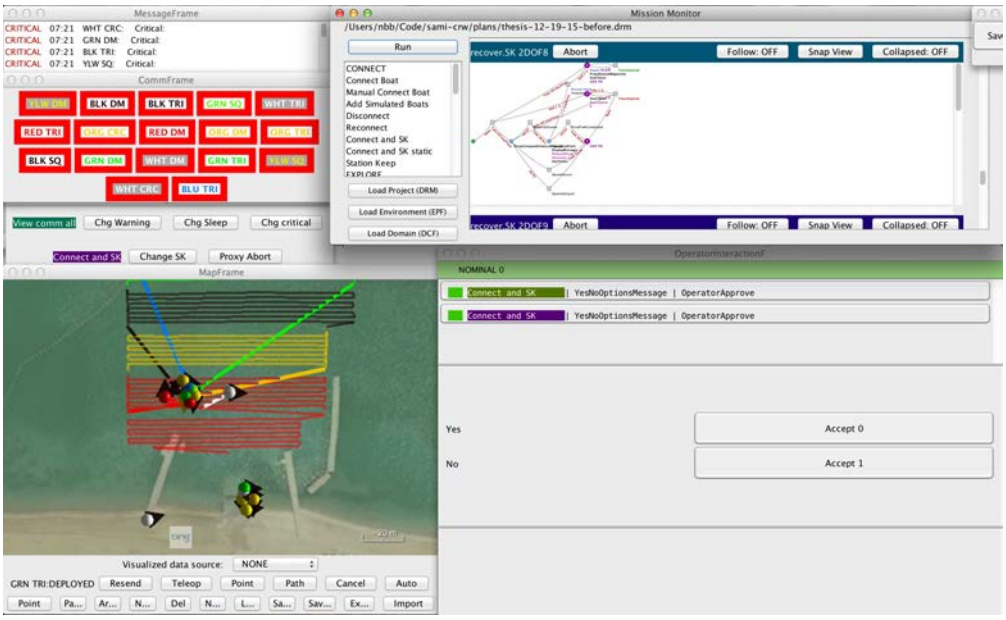
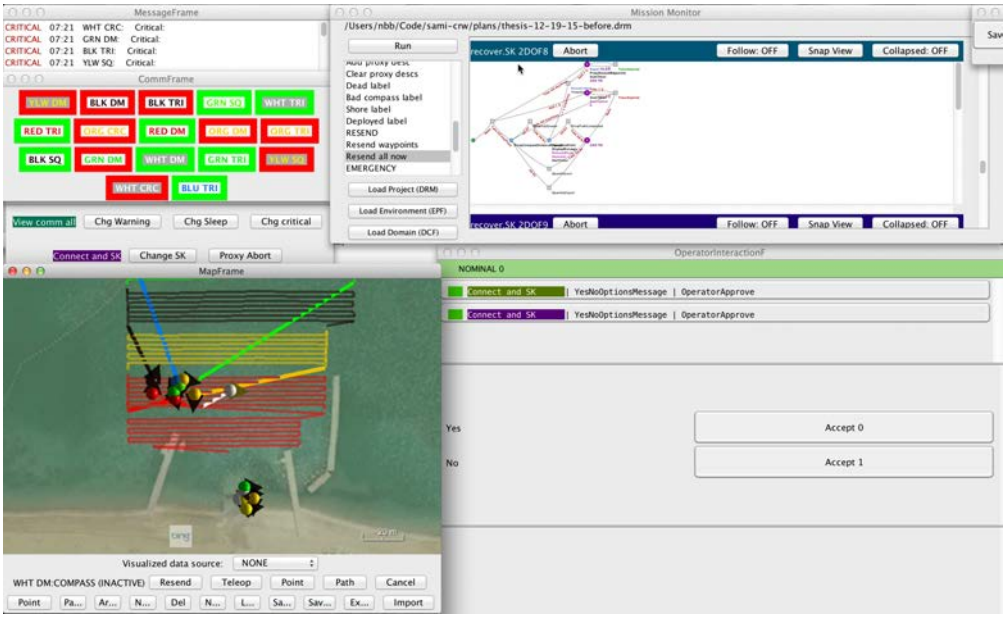


Figure 6.17: Communications Monitoring SPN



(a) Connectivity to all boats is poor



(b) Connectivity restored after network reconnection

Figure 6.18: Identifying WiFi network failure

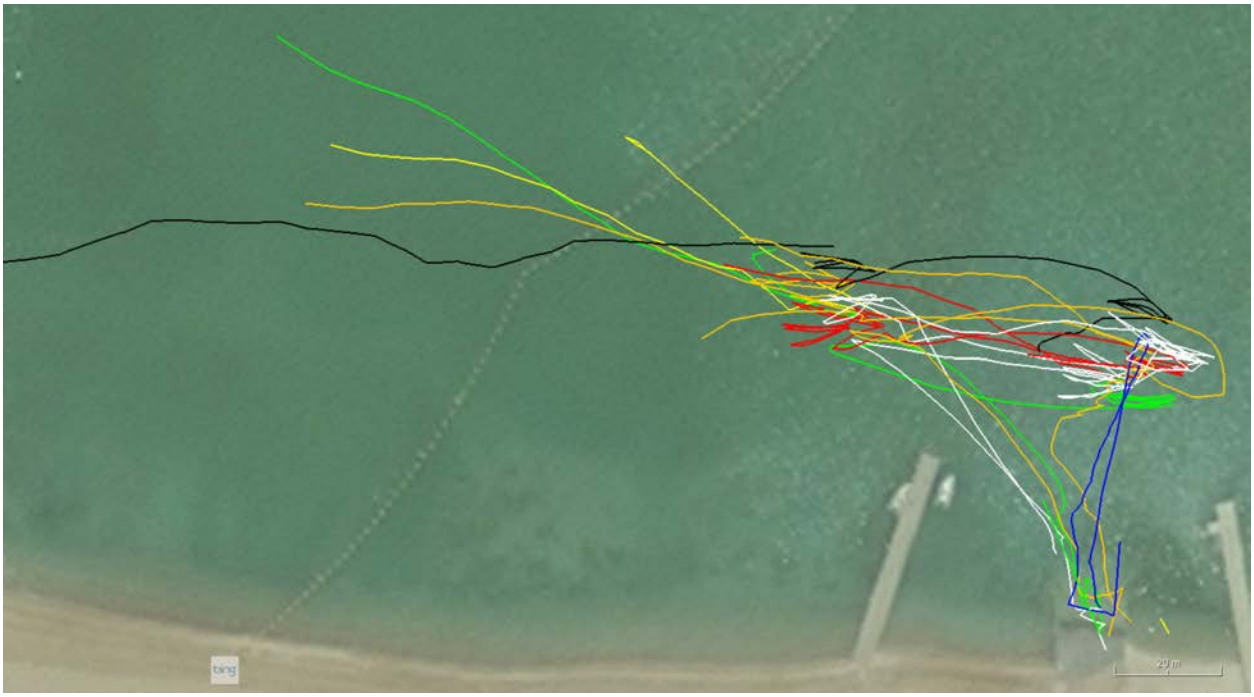


Figure 6.19: Boat paths during router failure

6.3.5 Communications Measurements

To begin thorough investigation of the relationship between phone height and wireless connectivity, two experiments were designed by a member of the CMU Qatar Networking Systems Lab. The goal of these experiments was to evaluate when positioning the phone differently could significantly improve connectivity or if additional hardware, such as external antennas for the boats, would be necessary. Ideally, a large, outdoor body of water with no current would be used to minimize boat motion while recreating environmental conditions. Such a body of water was not available, so these experiments were designed to be run at a field deployment location, where non-trivial water current is present.

The first experiment was designed to evaluate how changing the height of the phone relative to the water surface would affect connectivity between the phone and a laptop next to the base station antenna. Data packages were transferred between the laptop and a secondary phone attached to a propeller boat, recording transfer speed and number of dropped packets. Statistics were captured for several configurations varying the distance between the boat and base station antenna and the vertical mounting location of the experimental phone. To maintain a constant distance in a body of water while performing data transfer measurements, the boat was tethered to an anchor a desired distance from the base station antenna. The boat was then commanded to drive to a waypoint directly away from the

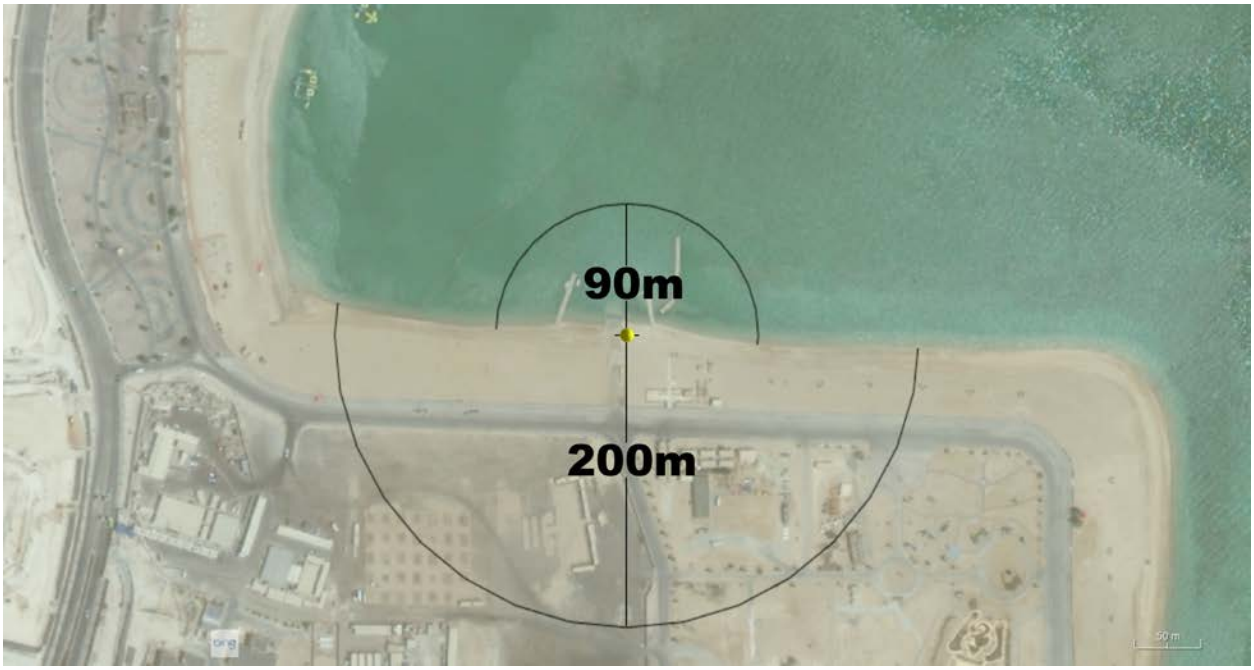


Figure 6.20: Distance (in meters) at which boats can maintain a stable connection to the base station antenna (yellow marker) while floating in the water (top semicircle) versus sitting on the shore (bottom semicircle)

antenna, introducing any motor electromagnetic interference and ensuring the phone was oriented in a consistent direction for the duration of the experiment.

The second experiment was designed to evaluate how changing the height of the phone relative to the water surface would affect connectivity between two phones in an ad-hoc network in addition to the relative heading between the two phones. A detailed text description and sketch, shown in Figure 6.21, of the desired motion of the boats which would fulfill desired experiment requirements were provided by a networking expert with limited knowledge of SPNs and other team planning languages. The description and sketch were then translated by an expert (myself) into a SPN. Figure 6.22 shows a subset of one version of this plan capturing boat movement for the first two waypoints. The plan begins with the operator selecting a boat twice. The first boat will be assigned the blue waypoints in the plan sketch, and the second the red waypoints. When a boat is selected, its token is moved either to the top half of the SPN, which will handle the blue waypoints, or the bottom half, which will handle the red waypoints. After both boats are selected the operator is asked to approve sending each boat their first waypoint. When the waypoints are approved and both are completed by the corresponding boat, the operator is again queried about starting the next set of waypoints. This allows the operator to run the necessary data collection and then trigger the next set of waypoints. Once the second set of waypoints is completed, the presented SPN ends.

The results of these experiments are not yet available, but its creation demonstrates the value in being able to rapidly prototype plans for non-robotics experts. The plan was able to be created in the field and iterated as runtime behavior was viewed.

6.3.6 Summary

In this chapter we presented lessons learned during hardware development, SPN plan design, and field deployment over several years of operating a fleet of autonomous boats. The fleet was operated in several environments with challenges ranging from quickly changing tides to motorboat traffic.

When implementing events for the first time in this boat domain, we were initially unsure as to the scope of an event. Initially, it was not clear if station keeping should be a single event or an entire plan. In general, it is simpler to err on the side of capturing the behavior with a single event. Once in the field it will quickly become apparent if the behavior requires more customization than a single event can provide.

Another observation during these deployments was the importance of rapid modification of team behavior to exploit or troubleshoot temporary phenomena in the environment or team. While we initially planned to spend most plan development time creating complex coordination, we found that reliable assistance with routine tasks, such as station keeping while incrementally deploying boats, was vital and more nuanced than anticipated. This suggests that building up a robust library of plans for routine actions should be the first

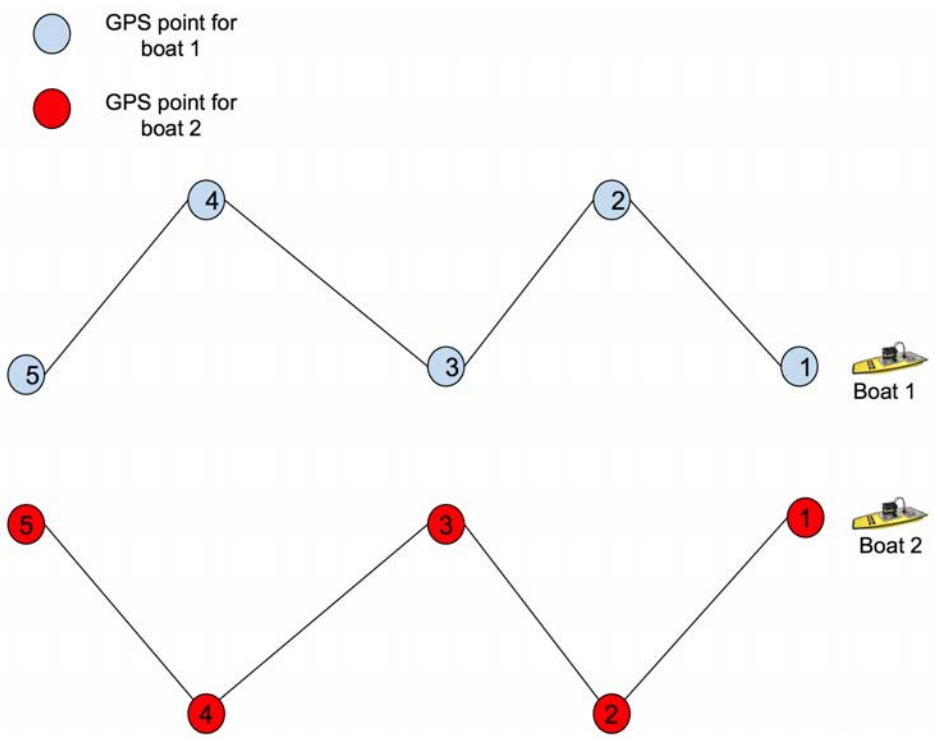


Figure 6.21: Non-expert sketch of a desired SPN

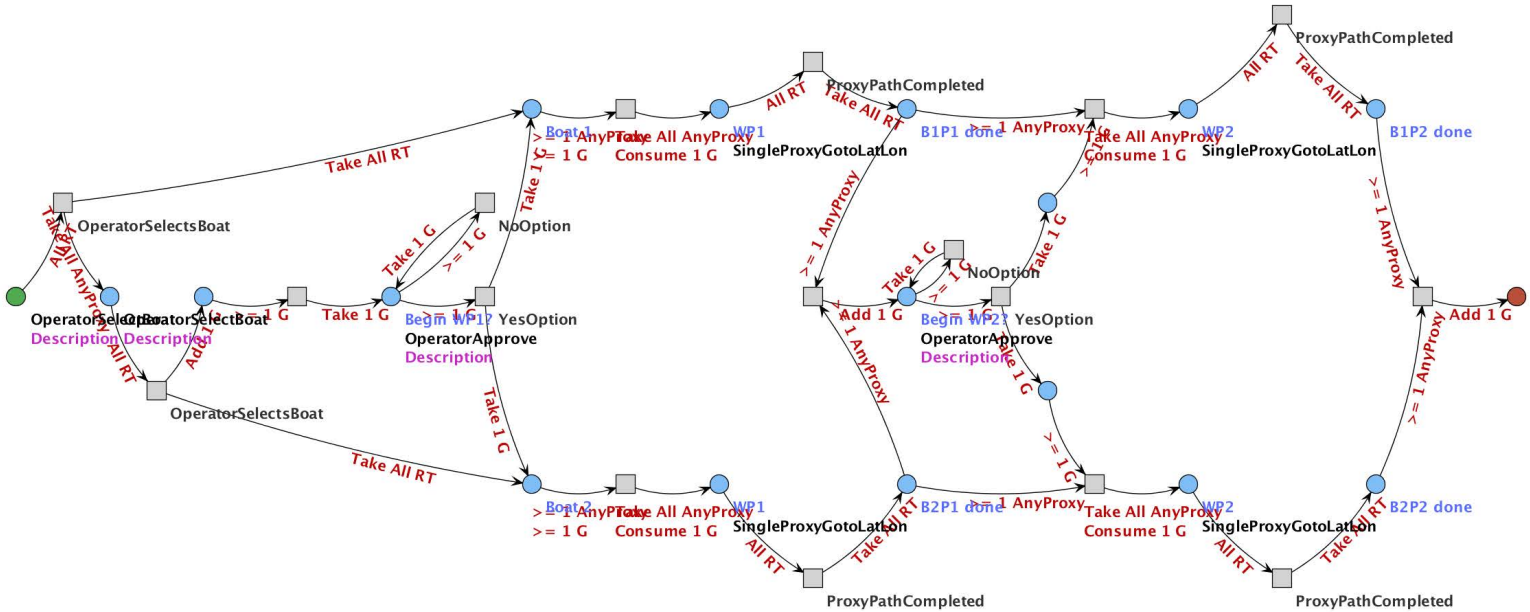


Figure 6.22: Two waypoint initial implementation of sketch

priority before developing more complex plans requiring a human expert's knowledge.

Chapter 7

Interrupt Evaluation

In this section, we describe the interrupt mechanism of SPNs following the syntax presented in Chapter 3. Afterwards we will discuss a "Cooperative Location Visit" SPN which makes use of interrupts [1].

7.0.1 PN Modeling of an Interrupt

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 [49]. Figure 7.1 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.

7.0.2 SPN Modeling of an Interrupt

Following the interrupt implementation idea described in Figure 7.1, we use three key elements to model the interrupt mechanism in the SPN framework: i) a place (called *Interrupt*

[1] This chapter is based on our JAAMAS 2016 article *Interacting with Team Oriented Plans in Multi-Robot Systems* [58].

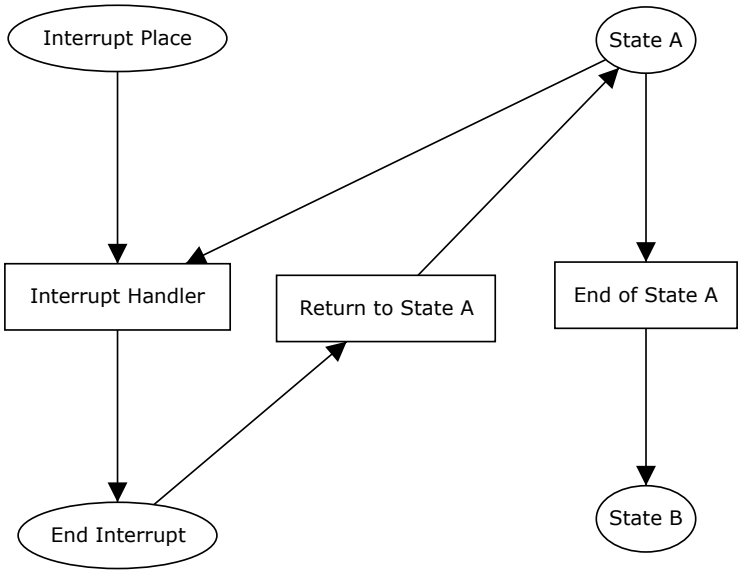


Figure 7.1: Interrupt implementation with Petri Net.

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).

Figure 7.2 shows the structures we propose to add interrupts to. We consider two types of interrupts: a *proxy* interrupt (see Figure 7.2a) and a *general* interrupt (see Figure 7.2b). 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[2]. 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.

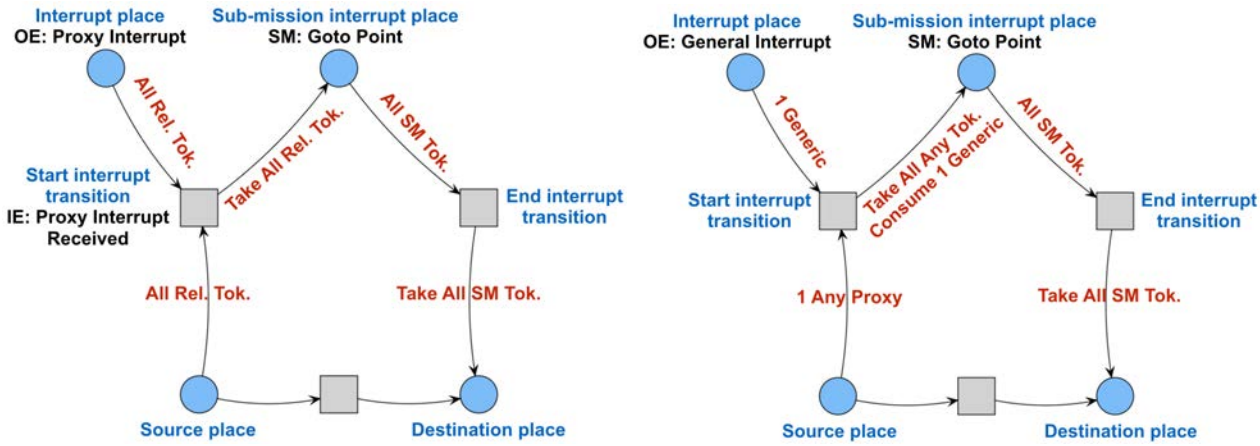
In contrast to the *proxy* interrupt, the *general* interrupt will remove all tokens present in the *Source place* and transfer them to the sub-mission. Hence, the event generated by the *Interrupt place* is a different output event, named *General Interrupt*. Such event is generated to trigger the interrupt 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.

7.0.3 Using Interrupts

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. For this scenario, the task allocation

[2]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 7.2: Types of interrupt implemented in the SPN framework.

method selected to assign boats to locations is based on Sequential Single Item auctions [184]. 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 [3].

The CLV plan is reported in Figure 7.3. 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 Chapter 3 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

[3]Since computing the minimum path cost given a sequence of visit locations is in general NP-Hard, we use a nearest neighbor heuristic: the path is built incrementally by selecting the location closest to the last added location for the boat. To select the first location, we choose the location closest to the boat's position.

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 7.0.2, 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 7.4 and discussed in detail in Section 7.1.

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.

7.1 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.

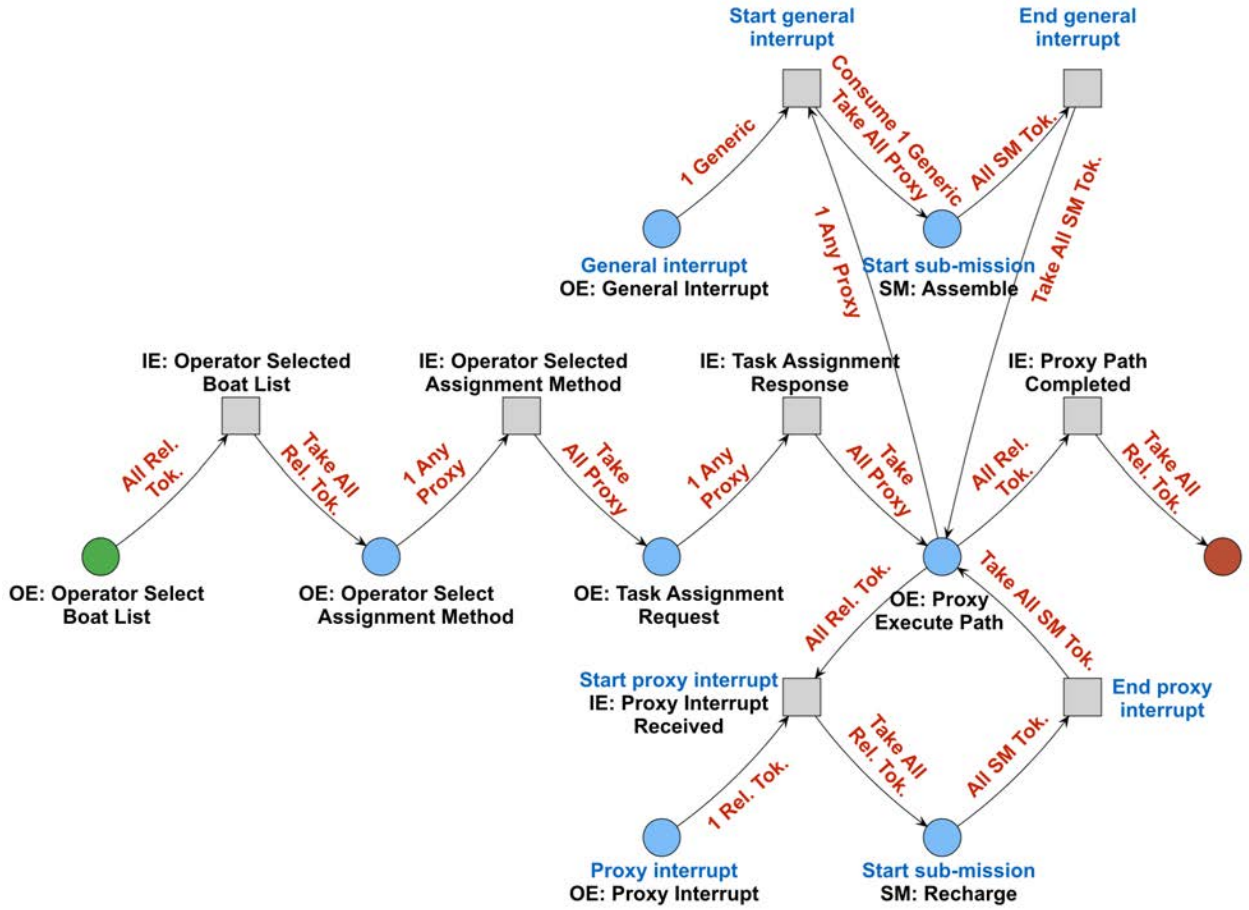


Figure 7.3: The Cooperative Location Visit plan specified in the SPN framework, with both general and proxy interrupts.

7.1.1 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 7.0.3: the "interrupt" version which encodes interrupts within the plan (reported in Figure 7.3) and the "standard" version without any interrupts (reported in Figure 3.1). 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 7.4 reports the CLV plan with a proxy interrupt to handle the traverse dangerous area incident. Specifically we report the parent plan in Figure 7.4a and the traverse dangerous area sub-mission plan in Figure 7.4b. Notice that in the parent plan (Figure 7.4a) 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 7.0.3 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). [4] Notice that the structure of the interrupt is the same as the one reported in Figure 7.2a, i.e., we have a place that

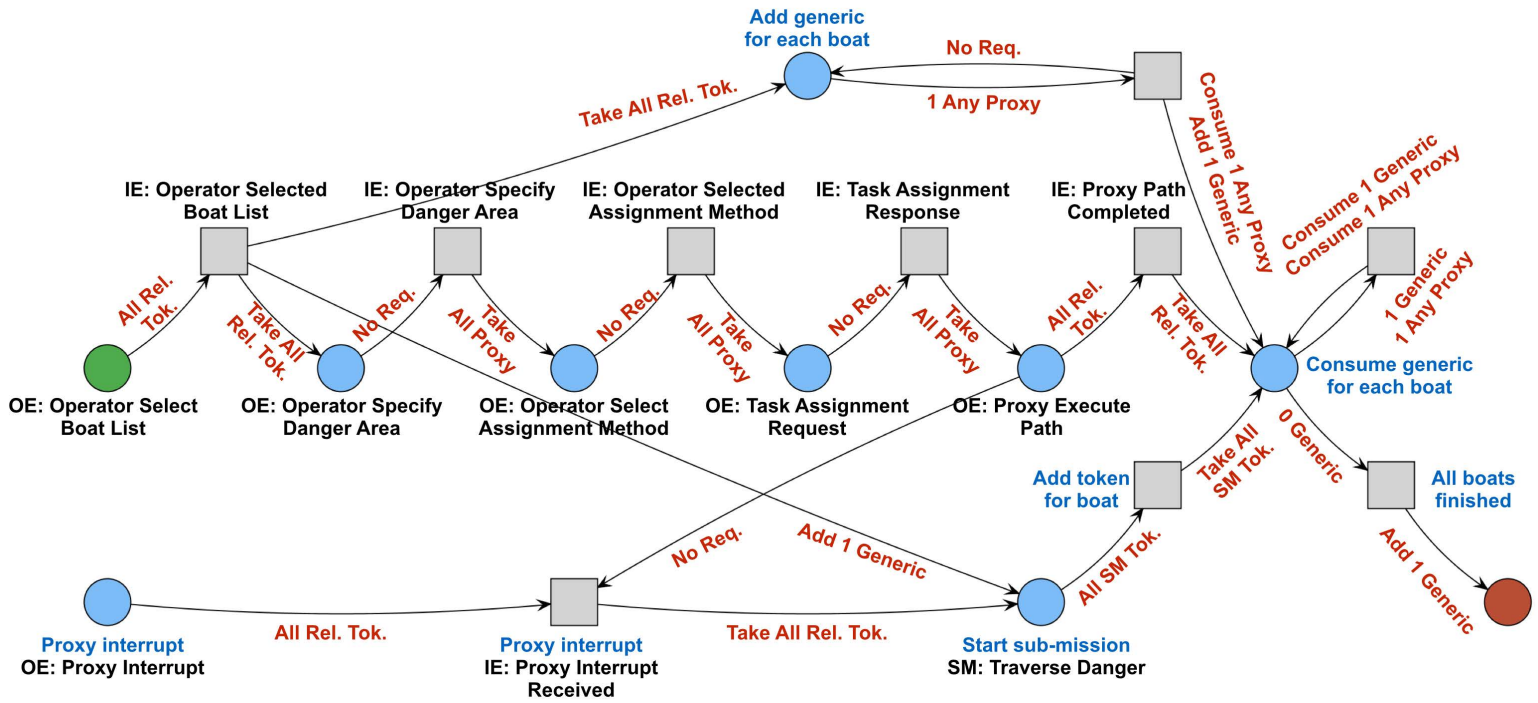
[4] While in our case the number of proxy / generic tokens is always finite, we might not know this number before the plan starts. Hence we use the inhibitor arc to check whether a place is empty.

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.

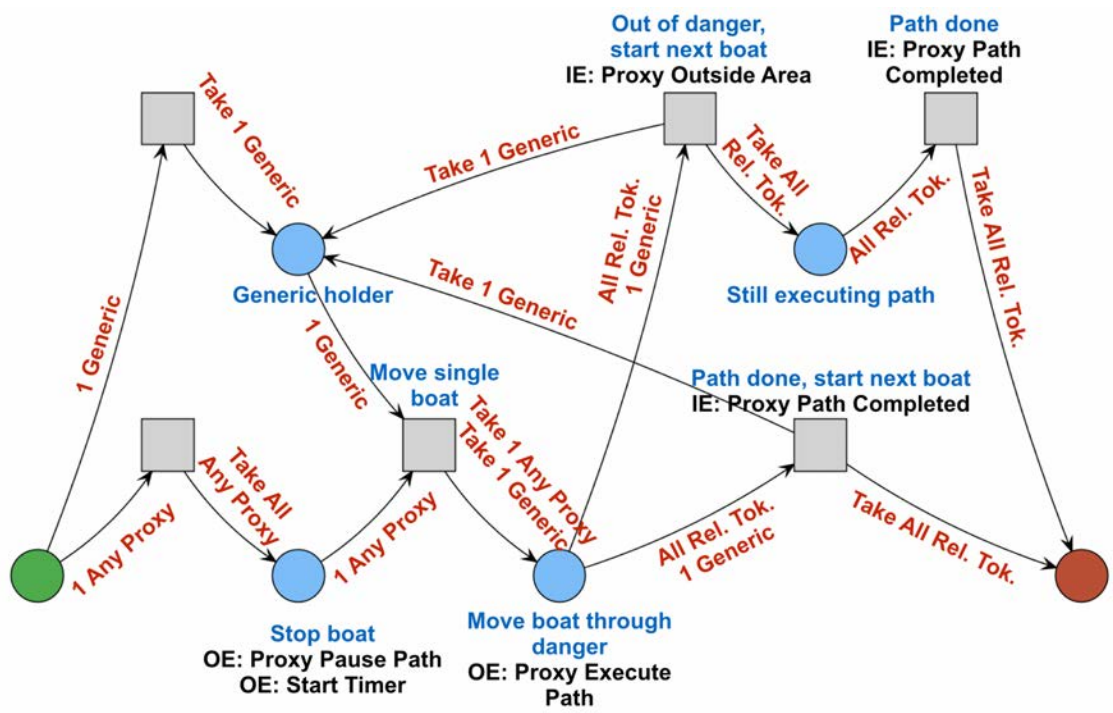
The "Traverse Dangerous Area" sub-mission reported in Figure 7.4b is used as static sub-mission (see Chapter 3) in the *Start sub-mission* place. Thus, when the transition holding the input event *Operator Selected Boat List* in the CLV plan fires, the generic token added to the *Start sub-mission* place 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 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



(a) The parent plan



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

Figure 7.4: CLV plan with the interrupt for traverse dangerous area

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

Configuration	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 7.1: 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.)

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 7.1).

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

7.1.2 Quantitative Results in Simulation

Table 7.1 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[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 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

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

Configuration	% 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 7.2: 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.

Configuration	% 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 7.3: 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
Configuration	% 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 7.4: 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.

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 7.2 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.

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 7.3 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 7.4 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 7.1, 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 7.1, while the third

column shows the results for number of interactions for the reassignment strategy version of the standard and interrupt plans[6]. 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 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 7.3 is reported here[7]. 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.

7.1.3 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[8] while Figure 7.5 reports a picture of the same run.

[6] 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.

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

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

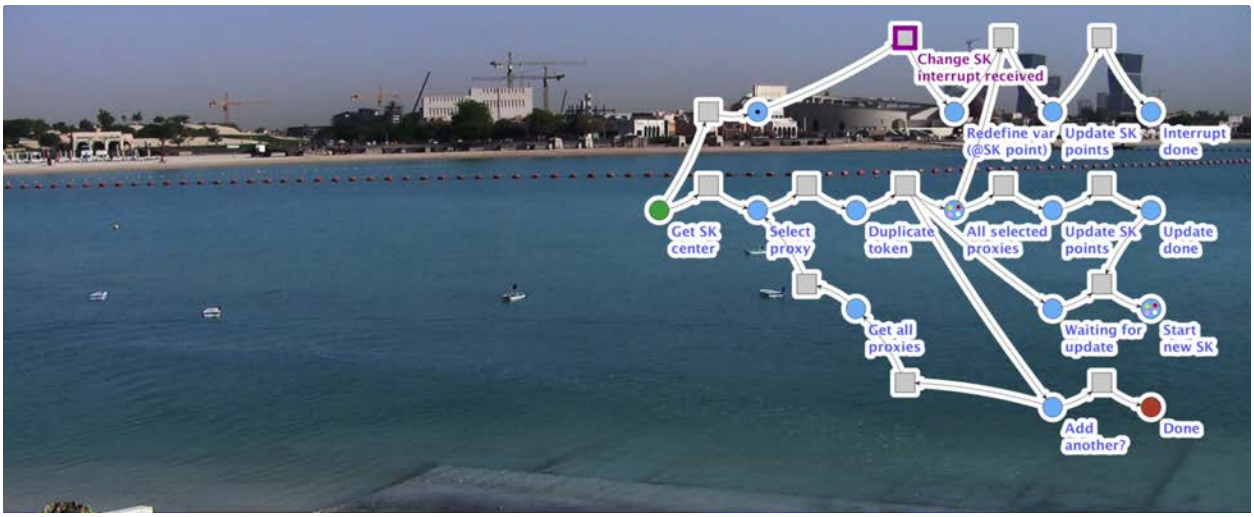


Figure 7.5: A picture of the connect and station keep experiment. The image shows a subset of the platforms and the current state of the SPN representing the connect and station keep plan. The interrupt portion of the plan is visible in the top part of the picture and the firing transition (highlighted in purple) is the one that starts the interrupt to change the position where boats should perform station keeping.

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). [9] This behavior is used in field deployments when large speed boats approach the current station keeping location, risking a collision with the robots.

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

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

7.1.4 Summary

In this chapter we discussed the motivation, implementation, simulated evaluation, and field operation of the SPN "interrupt" mechanism which allows an operator to quickly trigger complex behavior. Interrupts were a key feature of the language in field deployments and nearly all plans used in Chapter 6 had at least 2 interrupts. Support for globally scoped variables also proved to be important, as it allowed for information to be shared across plans and their interrupt behaviors. For instance, adjusting the "safe" recovery location in one plan due to receded tides or a newly docked boat could then be carried across to other plans.

Chapter 8

Markup Evaluation

In this chapter we present an experiment to evaluate markup performance.

The unpredictability of field deployments makes controlled experiments extremely difficult. Instead, we will use the Cooperative Robotic Watercraft project simulator to compare performance of a plan with SAMI markup to a plan without SAMI markup. The simulation described below is designed based on observations from field deployments described in Chapter 6. By changing the number of robots, supporting team members, robot hardware, GUI, operator skill level, location, or many other factors, the behavior of the team can vary drastically. This limits the usefulness of an evaluation based on absolute measurements, so we focus evaluation on trend based metrics.

8.1 Design

Two versions of the plan were compared: a *markup* version with SAMI markup customized for each event and a *generic* version with identical markup across similar events. The latter case represents a team plan in a framework without SAMI markup, where the UI, robots, and AI treat each instance of an event class identically.

In the scenario, a team of boats is used to perform sensor mapping in the area of a dock. Occasionally, manned boats will enter or leave their section of the dock. If the operator notices the boat, they will use an interrupt to enable a section of the SPN where they specify a safe location for the robots to temporarily station keep at while the boat enters or leaves the area. If the operator fails to notice the boat, or the robots don't move fast enough, the robots could get damaged by a collision with the much larger manned boat. In the version of the plan with customized markup, movement events in the interrupt section of the plan will have markup specifying speed should be optimized. This will result in robots moving twice as fast when performing interrupt-related movement in the markup plan. In the generic plan, robots will move at the same base velocity whether they are mapping or

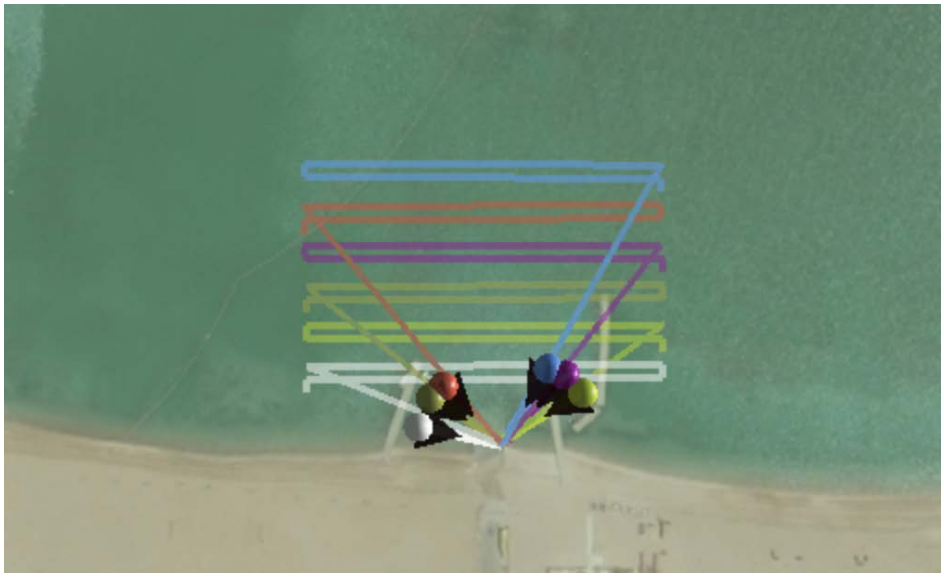


Figure 8.1: Avoiding a hazard while mapping an area

avoiding a boat.

The team consists of simulated robots with properties modeling those of the Lutra propeller boat described in Chapter 6 and a simulated operator. As described in Chapter 5, the operator has a prioritized queue of decisions which is populated by SPNs. The operator is simulated by consuming the front item of this queue after waiting for a period of time approximating the decision time for a human operator. To simulate the complexity of a live field deployment, in which the operator's attention is also required to make observations about the team, "background" actions are also added to the decision queue, including:

- Scanning area for new hazards
- Checking each robot for anomalous behavior
- Actions to prepare other robots for deployment
- Interpreting received data

Scanning area for new hazards may reveal static obstacles, such as submerged trees or buoys, or dynamic obstacles, such as speedboats. If an obstacle is seen measures are taken to avoid it, such as adding the obstacle to the path planner via the GUI, running a separate SPN to map and add the periphery of the object to the path planner, and triggering an interrupt to move out of a dynamic obstacles path. In this scenario, the operator spends 30 second segments looking for incoming watercraft. If an obstacle is detected, an interrupt is invoked which will move the robots to a pre-designated safe location. After arriving,

the boats wait 30 seconds to allow the watercraft to pass without harming the boats, then resume their previous actions. Figure 8.1 shows a team of 6 simulated robots moving to a safe location when their exploration is interrupted.

Checking robots for unexpected behavior consists of checking the GUI for warnings, such as a low battery, and watching visible robots' behavior to identify failures the software cannot recognize, such as a localization failure. Methods for addressing this behavior include teleoperation, running a separate diagnostic SPN, and triggering an interrupt. In this scenario, the operator spends 30 second segments looking for anomalous behavior.

Actions to prep other robots for deployment consists of assembling components, starting software, checking diagnostics, and moving prepared robots into the field. In this scenario, the operator spends 30 second segments performing tasks and coordinating with other human team members to prepare or repair other robots.

Interpreting received data consists of reviewing sensor data and progress in the SPN and comparing it to previous data and the operator's expectations Unexpected data may result in running additional plans, discussing progress with other team members, or making notes for post-deployment analysis. In this scenario, the operator spends 30 second segments reviewing received data and monitoring progress.

8.2 Results

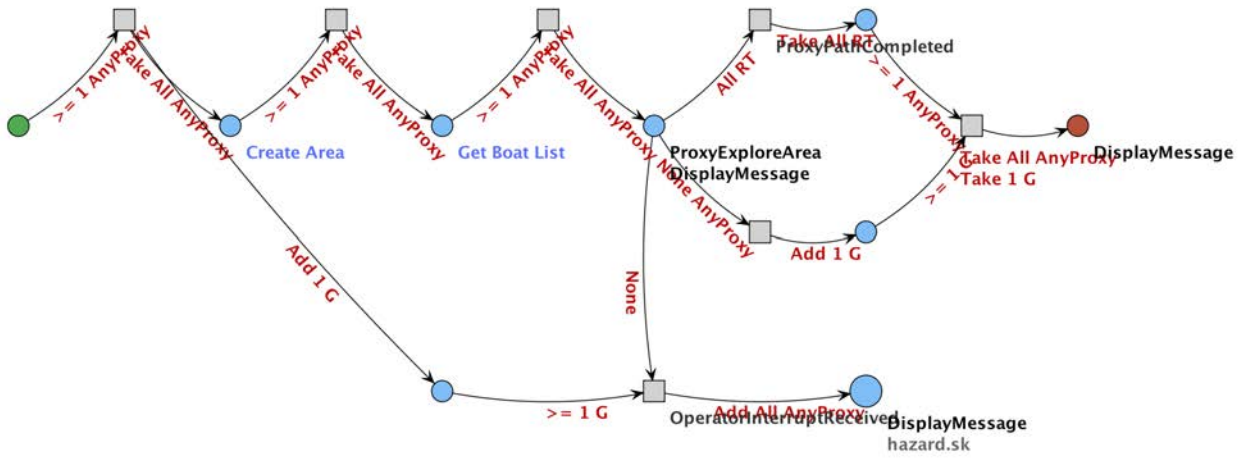
The simulation results when running the two versions of the SPN were recorded for several different team sizes and the following metrics were compared

- Mission time: The time it takes for the exploration SPN to be completed
- Hazard resolution time (seconds): The time it takes the robots to move from their positions when the interrupt is triggered to the safe location
- Number of measurements: Number of *unique* measurements taken, where unique means no other measurements have been taken within 1m of this measurement

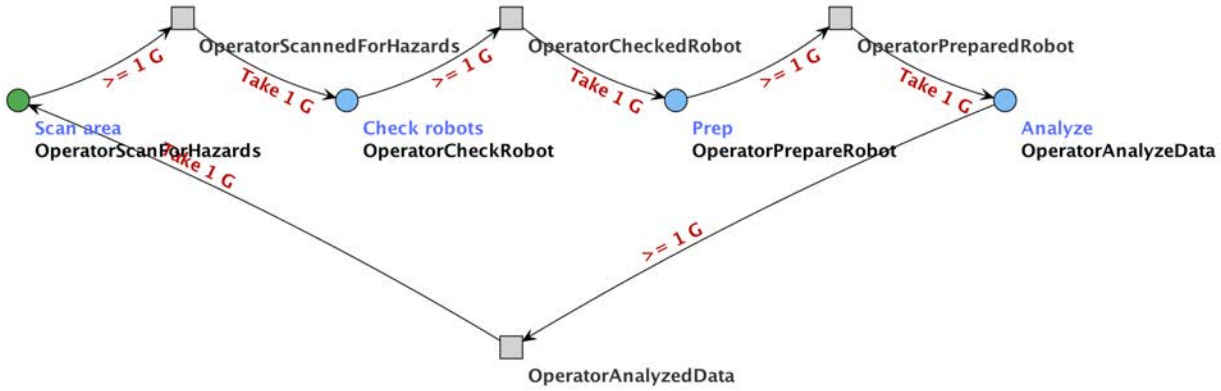
Table 8.1 shows the results from the simulation and the percent gain of the markup version (M) of the plan compared to the generic version (G). Percent gain is calculated using the formula

$$\%Gain = \frac{quantity_{markup} - quantity_{generic}}{quantity_{generic}} \times 100$$

For mission time and hazard resolution time, we want a decrease (negative gain). For number of measurements, we want an increase (positive gain). The hazard resolution time decreases substantially in the markup version of the plan. As the robot speed is doubled during hazard resolution in the markup version, it is expected the reduction would be approximately



(a) Scenario 1 SPN



(b) SPN for injecting simulated operator background actions

Figure 8.2: SPNs used during markup evaluation simulation

Performance metric	Number of robots			
	3	4	5	6
G Hazard resolution time (s)	32	32	33	32
M Hazard resolution time (s)	17	17	17	17
% Gain (-)	-47	-47	-48	-47
G Mission time (s)	515	435	368	303
M Mission time (s)	500	419	352	287
% Gain (-)	-2.9	-3.7	-4.3	-5.3
G Number of measurements	2679	2756	2714	2708
M Number of measurements	2635	2740	2699	2692
% Gain (+)	-1.6	-0.58	-0.55	-0.59

Table 8.1: Percent gain for statistics comparing markup (M) version of plan to generic (G) version of plan. (-) indicates a negative value is desired, (+) indicates a positive number is desired.

50%. By reducing the hazard resolution time, the mission time is also reduced, resulting in short mission times for the markup plan. Furthermore, as the time taken to explore the area decreases as team size increases, the fraction of mission time corresponding to hazard resolution also increases. This results in an increasingly negative percent gain as the team size increases. However, the number of measurements taken decreases for the markup version of the plan. This is due to the sampling rate of the simulated sensor. When the team is moving from their mapping area to the safe area, they continue collecting measurements. In the markup version of the plan, the faster movement rate results in fewer measurements being taken. However, this does not have a large impact on the overall number of measurements as the explored region is large. The number of robots does not appear to significantly affect the percent gain. This is likely due to other factors which also affect the number of measurements taken when reacting to a hazard, such as distance from and angle to the safe location.

8.2.1 Summary

In this chapter we presented evolution of the markup language over several years of field deployments and a simulation evaluating of the effects of SAMI markup on SPN plan performance. Markup was applied to a boat exploration plan to prioritize resolution of hazard avoidance in the plan over other factors such as measurement quantity.

As the motivation for markup requires situations where context demands a change in system behavior, it can be difficult to evaluate its expressiveness through simulations which are sufficiently nuanced without being overly engineered. In addition to quantitative performance evaluation, it is also important to simply practice writing marked up plans for

hypothetical scenarios in a familiar domain and scenarios presented in related literature in order to identify limitations of the current markup vocabulary.

Chapter 9

User Study

An important step in making robot teams common tools in the real world is improving their accessibility. The accessibility of current systems is largely limited by the need for a robotics or planning expert to write and adjust plans each time goals or scenarios change. However, basic programming skills are becoming widespread in the scientific community for automating tasks, designing simulations, and analyzing results. As a result, in many of the domains where robot teams offer great potential there will be user(s) with a basic programming background. If training for using a team planning language builds naturally off of basic programming concepts, these users may be able to learn and apply it to their domain [57, 144, 181, 77, 139, 143, 78]. However, if the training takes more time than would be saved by using the robot team, then there is less incentive. Similar to how the accessibility of robot teams is limited by the need for an planning expert, the scalability of the training process should not be limited by requiring the presence of a language expert.

In this chapter, we will discuss the design and evaluation of training material designed to teach non-experts to use the SPN language with 4 goals in mind:

- Identify the range of user backgrounds the training material is appropriate for
- Create training material that can be scaled to large numbers of users
- Measure how well users understand SPN syntax after completing the training material
- Measure how well users can transfer knowledge to actual scenarios after completing the training material

9.1 Study Design

9.1.1 Capturing the Demographic

One of the goals of this thesis is to provide a team planning language useable by non-experts. Our concept of a non-expert is based on two factors. First, a non-expert would have some knowledge of programming, but programming would not their field of expertise. Second, a non-expert may have some knowledge of planning or petri nets, but again would not be their field of expertise.

For our purposes, we consider non-expert programmers as those with some knowledge of programming, but programming is not their field of expertise. To measure programming knowledge, we consider relationships between programming expertise, completed degrees, and degrees in progress. Degrees in computer science generally involve a high level of programming. Degrees in engineering generally require some programming. Those currently pursuing a degree have not yet been exposed to all the material required to complete the degree, so we consider completed degrees to yield more programming expertise. In addition we consider self-reported frequency of programming and familiar programming languages. We anticipate that users who program on a daily basis will in general have more expertise than those who program less frequently. We also consider the number and classification of familiar programming languages. We hypothesize that more languages and a variety of types of languages translates to more programming expertise. In addition, experience with certain programming languages may translate to better learning of the SPN language than others. For example, the R programming language is primarily used for statistical computation, whereas C++ is commonly used for applications requiring logical operations common in team plans.

We consider two factors when evaluating background with planning problems and languages. Similar to programming experience, we consider educational background. Those currently pursuing a degree in robotics or having completed a degree in robotics will likely have some exposure to planning problems. In additional to educational background, we consider self-reported familiarity with petri nets. We consider knowledge of Petri Nets to be uncommon and that existing experience would indicate an expert background in planning. Furthermore, those with knowledge of Petri Nets will already be familiar with much of the lesson content and will be less effective at evaluating the efficacy of the training material.

The resulting survey capturing these attributes is shown in Figure 9.1. Participants were recruited through a university participant pool mailing list and bulletin board advertisements in engineering and computer science buildings. These advertisements directed interested viewers to fill out the recruitment survey and, if selected, to fill out a schedule of available times.

When selecting the participant, we excluded applicants considered to have novice or expert knowledge of the above areas. For our purposes, we considered an applicant to be an

expert if **any** of the following were true:

- Have completed a degree in computer science or robotics **and** program on a daily basis
- Have used Petri Nets

We considered an applicant to be a novice if the following was true:

- Have not completed a degree in computer science **and** do not currently program **and** have not used Petri Nets

Users who were neither an expert nor novice were classified as non-experts.

9.1.2 Designing the Material

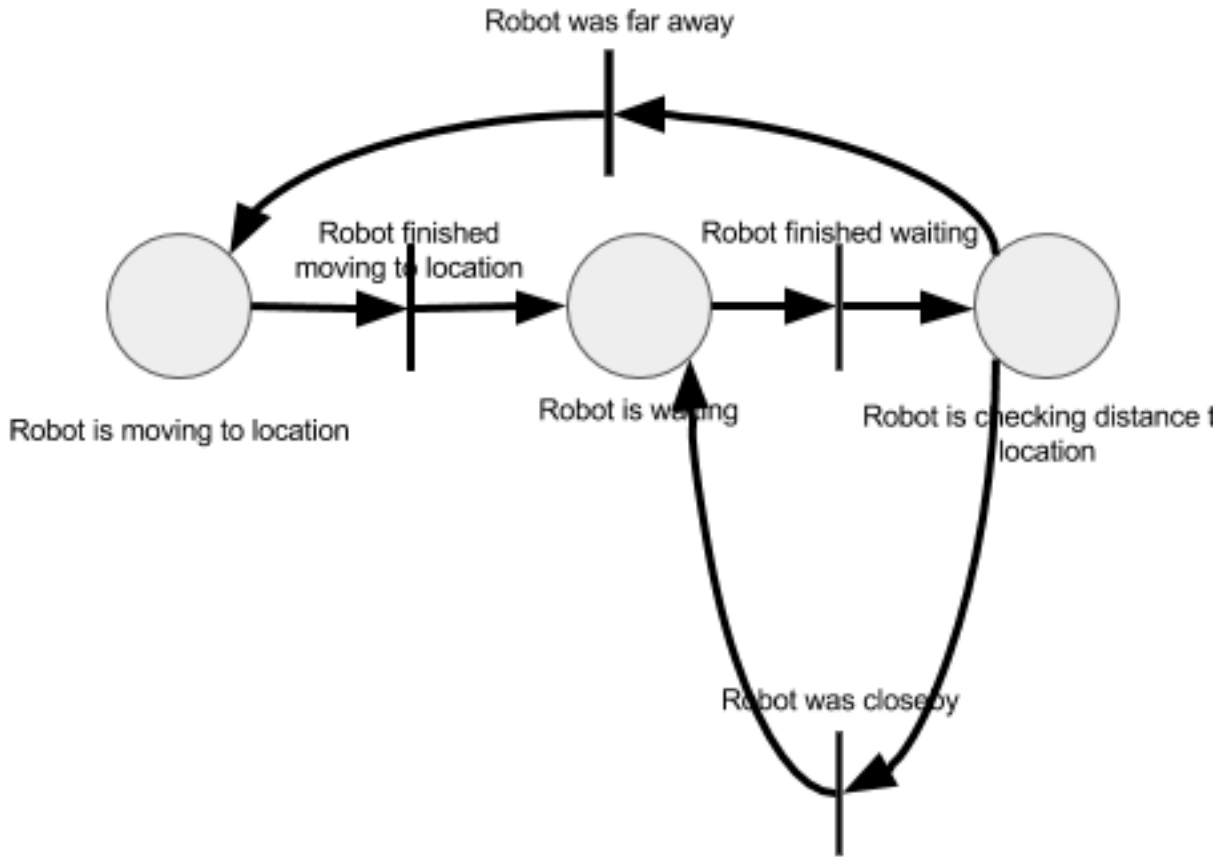
Lessons were designed with the ultimate goal of being packaged with the SPN software and being completely self guided. The lessons used a slideshow format and incrementally introduced new syntax or capabilities in the language. This concept of *incremental complexity* was an important step in reducing the size and duration of the training material. By choosing the scope of the entirety of the training material before developing individual lesson content, it was possible to choose a total of 2 SPN team plans which could be incrementally built up through the individual lessons. While this requires more planning when designing lesson content, it avoids a new team plan being introduced each lesson, which increases lesson size and duration. Figure 9.2a shows the first iteration of "Connect and Station Keep" plan, used in Lesson 2, and Figure 9.2b shows the seventh and final iteration of the plan, used in Lesson 8. The tradeoff in this approach is that future expansion of the training material may require reworking some or all of the existing lessons to retain a small number of team plans. The slideshow format was selected as it allows for easy distribution online, a self-guided pace, and on-demand review of earlier concepts. Lesson slides occasionally included captioned videos demonstrating how to perform a specific action in DREAMM, such as adding a place or changing a variable's value. Periodically slides contained a *Quiz* or a *Job* for the user used to assess syntax understanding and knowledge transfer, respectively.

Quizzes presented sample SPNs and asked the user questions about its state or behavior based on syntax rules that had been presented. After completing each quiz, the solution was immediately presented to the user so they could identify any sources of misunderstanding. Answering quizzes generally involved multiple choice or selecting elements in a presented SPN matching specified requirements. This will simplify notifying users of errors in a fully self-guided version of the training. In addition, it will allow for identifying quizzes which are commonly answered incorrectly and may need clarification or more supporting content.

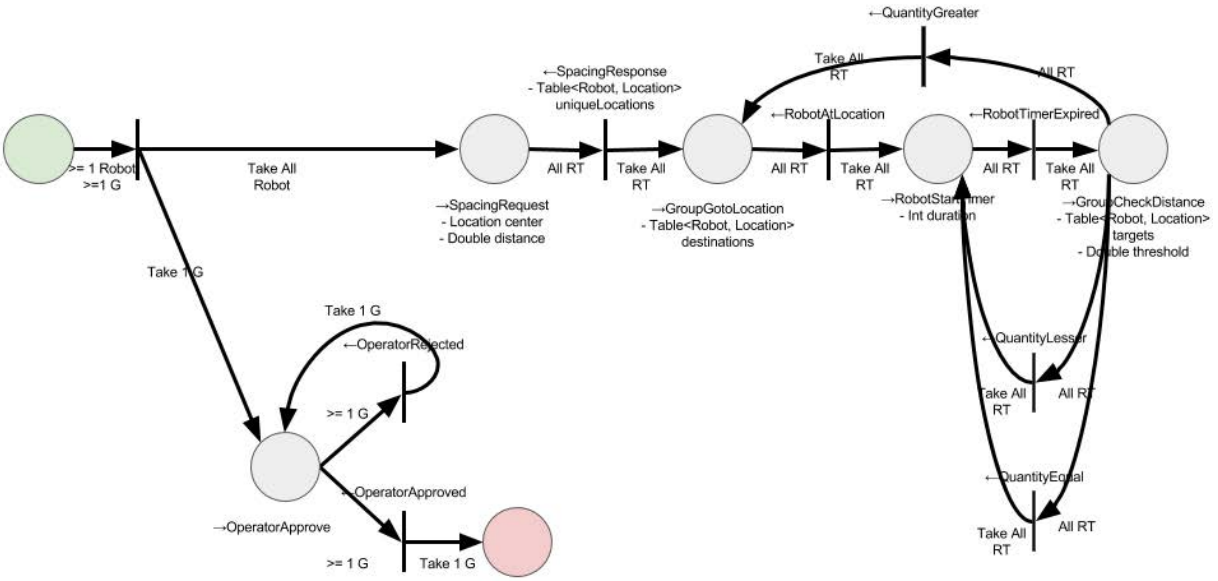
Jobs instructed users to apply their knowledge to modify SPNs in DREAMM to include newly presented syntax rules or to achieve a new goal. To reduce training time and test understanding of the language rather than power of language tools, DREAMM assistant

1. Name:
2. Age:
3. What is your affiliation with CMU? (select one)
   - Student
   - Faculty
   - Staff
   - Other
4. List all degrees in progress:
5. List all completed degrees:
6. Which best describes your programming background? (select one)
   - Daily
   - Weekly
   - Monthly
   - I do not currently program
7. List all programming languages you are comfortable programming in:
8. Which best describes your knowledge of Petri Nets? (select one)
   - I have not heard of Petri Nets
   - I have used Petri Nets before
   - I currently use Petri Nets for some task

Figure 9.1: Recruitment Survey



(a) First iteration



(b) Seventh (final) iteration

Figure 9.2: Incremental complexity in Connect and Station Keep SPN

agents developed using properties discussed in Chapter 4 were disabled. A simple agent which added a graphical reminder to edges without edge requirements was enabled, but did not provide suggestions for edge requirements. A future user study could compare the effectiveness and user preferences concerning assistant agents. To reduce user confusion, the DREAMM editor was modified to support *lesson modes* corresponding to each lesson: when in a lesson mode, elements of the language or GUI which had not yet presented to the user were hidden. To reduce user confusion, the DREAMM editor was modified to support *lesson modes* corresponding to each lesson: when in a lesson mode, elements of the language or GUI which had not yet presented to the user were hidden. For example, before edge requirements are introduced, the only option when clicking on an edge is to delete it, and compilation warnings due to unspecified edge requirements are suppressed. Section .4.1 provides more detail about these lesson modes. Additionally, lesson specific domain configuration files were used to ensure that only events and task classes which had been introduced to the user were presented as options in the corresponding dialog boxes. Jobs were designed to generally require only a few actions and have a limited number of correct variations so that evaluating the user's answer could be automated for the most common answers.

Lesson List

In total, 10 lessons and a *final plan* were designed. A rough description of each lesson is provided below.

Lesson 1: Motivation: What are team plans and why do we use them?
Lesson 2: Introduction: Introduction of places and transitions
Lesson 3: Events: Making team members do things via events
Lesson 4: Generic tokens: Using the concept of a token to represent a single robot's status
Lesson 5: Edge requirements: Moving a robot's token around the SPN to in response to events
Lesson 6: Robot tokens: Using colored tokens to represent multiple robots
Lesson 7: Operator interaction: Getting information from the operator at run-time
Lesson 8: Variables: Storing and retrieving information sent by team members
Lesson 9: Tasks and task tokens: Creating, allocating, and performing tasks
Lesson 10: SAMI markup: Changing mixed initiative autonomy and GUI presentation
Final plan construction: Constructing a SPN from a text description

Additional lessons were developed for distribution to collaborators, but were not included in the user study to reduce experiment duration and increase participant retention.
Lesson 11: Sub-missions: How to re-use common SPN sections as sub-missions
Lesson 12: Contingencies: How to set up portions of a SPN to trigger based on a received

input event indicating a contingency measure is needed
Lesson 13: Interrupts: How to set up portions of a SPN to trigger based on a received interrupt from the operator

Some slides from Lesson 6 on robot tokens are shown in Figure 9.3. The slides for each lesson are provided in Appendix .4.2. The lessons were built around two motivating plans: (1) station keeping (Figure 9.4) and (2) measurement collection (Figure 9.5). Users slowly built up to the final station keeping plan in lessons 1 through 8. Initially the station keeping plan was built for one robot with no concept of token color or termination condition. As the idea of multiple robots and an operator were introduced in subsequent lessons, the station keeping plan was expanded and modified to incorporate those concepts. Similarly, users built up the measurement collection plan in lessons 9 and 10 beginning with a simple task allocation plan.

Final Plan

In the final plan, participants were instructed to design a SPN which would achieve the following:

- First, the operator should be asked to create a list of locations
- When the list of locations is received, Camera tasks should be generated from the list of locations
- When the task tokens are received, they should be allocated immediately by the System AI.
- When a task is assigned, the robot for the task should move directly to the tasks location
- When the robot for the task arrives, it should take a panorama
- After taking the picture, the robot should wait 5 seconds
- After waiting 5 seconds, the task is complete
- When there are no unfinished tasks, the mission should end

Figure 9.6 shows the initial version of the final plan participants were instructed to create. Similar to lesson jobs, the role of the final plan was to help asses how well the user could transfer their knowledge of the language into a SPN. After completing the initial version of the plan, participants were instructed to modify it such that the panoramas would be taken simultaneously, with the assumption that the tasks could all be allocated at the same time

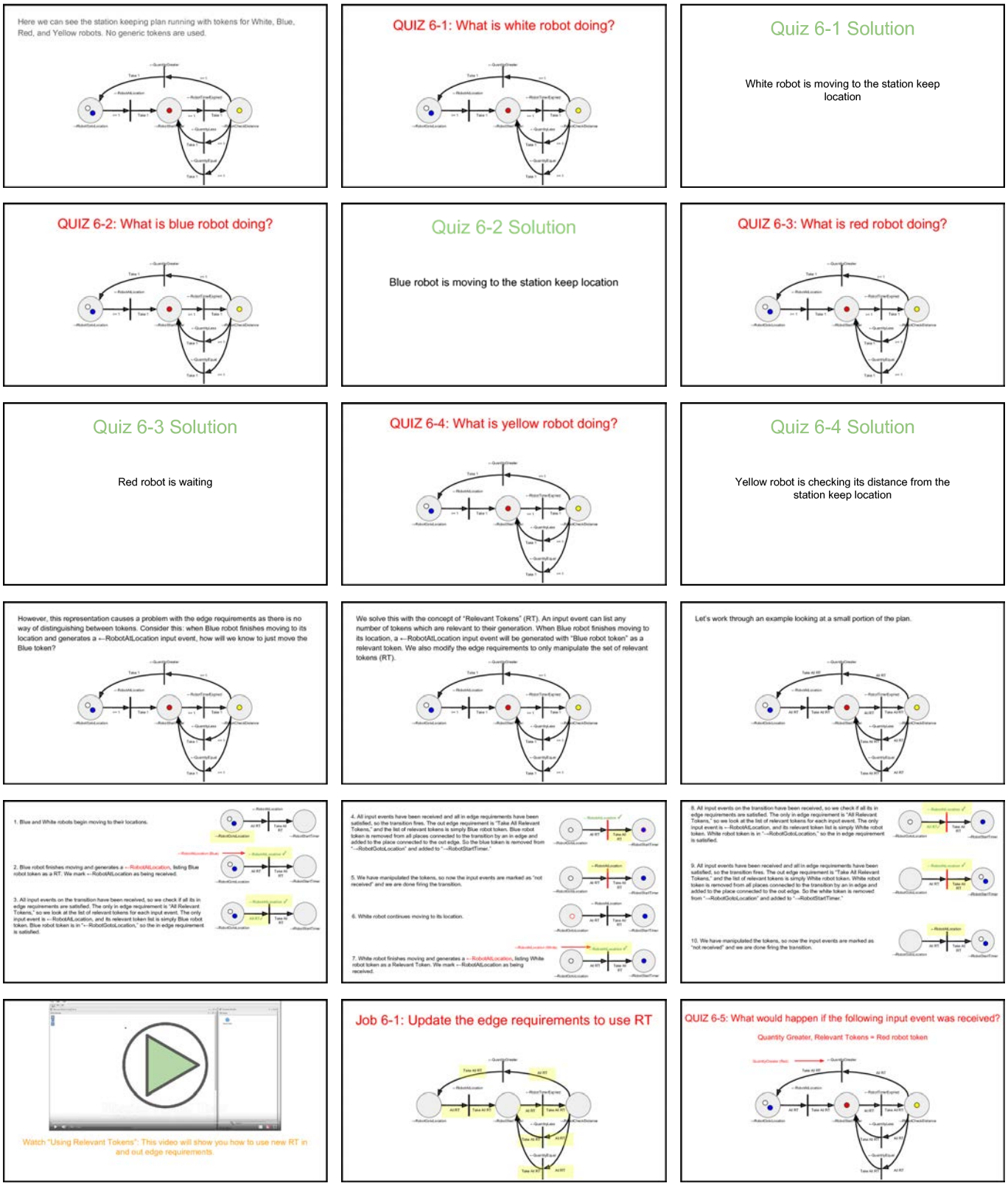


Figure 9.3: Some slides from Lesson 6: Robot tokens

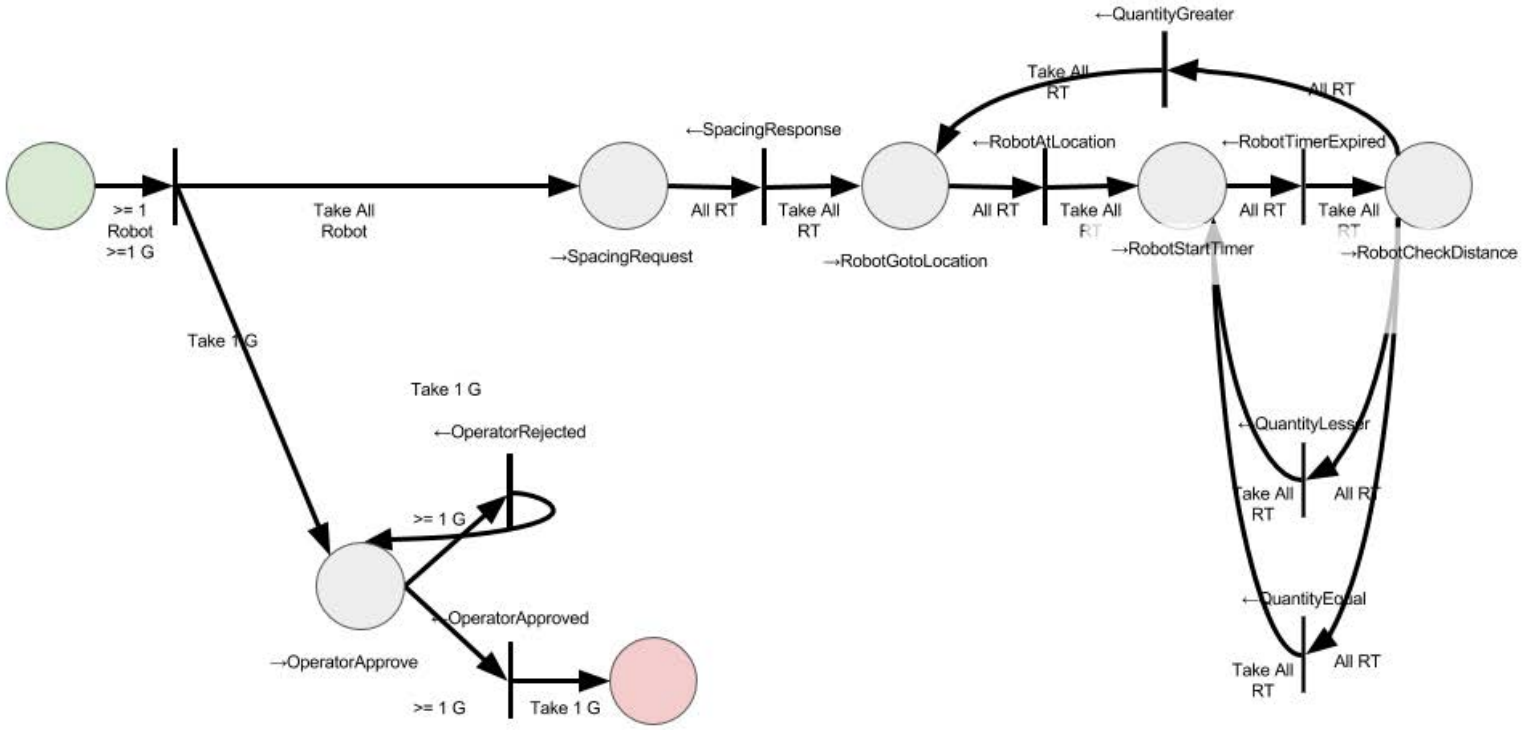


Figure 9.4: Station keeping SPN

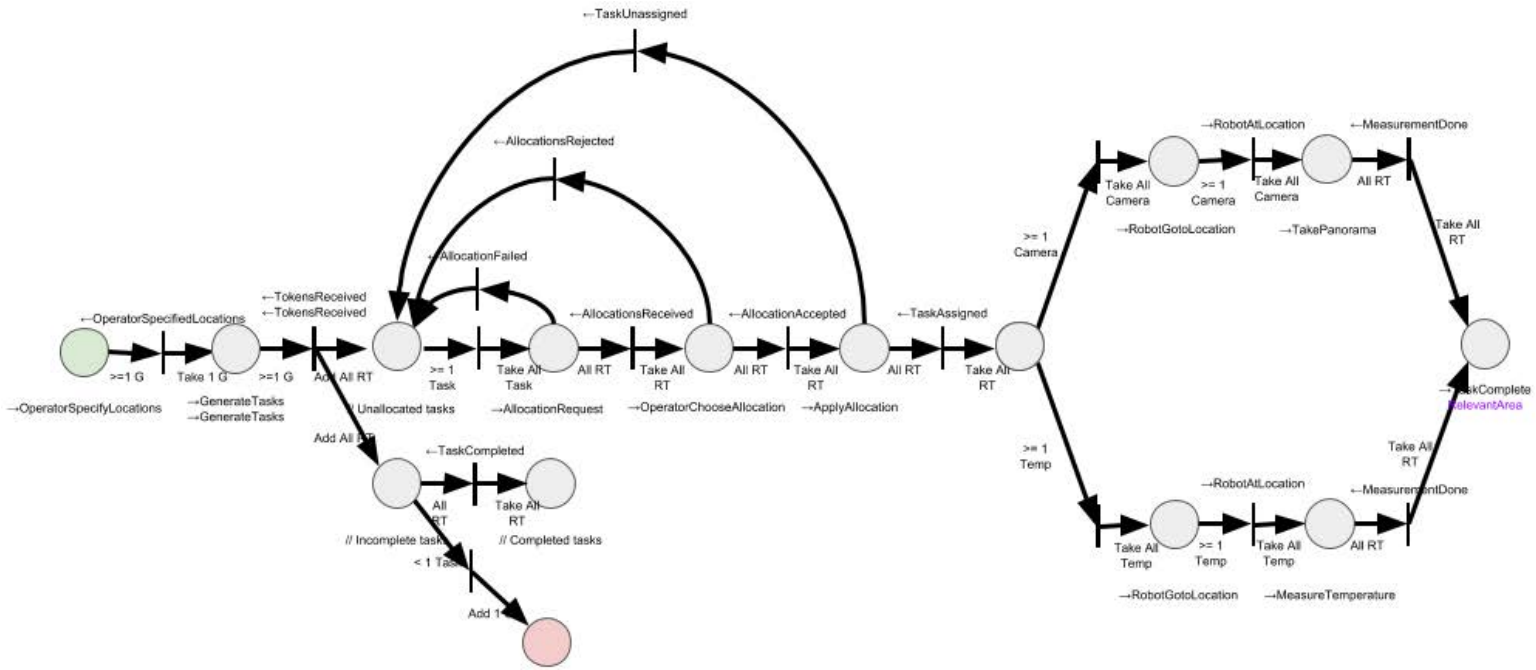


Figure 9.5: Measurement collection SPN

(i.e., there were at least as many boats with cameras as there were camera tasks). Figure 9.7 shows a modified version of the final plan achieving this goal. The completed lessons did not demonstrate how to achieve this effect, though a similar mechanism was used in a lesson to end a plan when all tasks were completed. It was our hypothesis that users would not be able to fully complete this task. If users were able to complete this task easily, it would be evidence that the lessons could be more abstract and thus shorter.

Cognitive Walkthrough

To improve learnability, a cognitive walkthrough [196, 176] approach was used when designing quizzes and jobs. In a cognitive walkthrough, evaluator(s) work through tasks being designed from the perspective of the target user. The goal is to tell a story for each action in a sequence required to perform a task, taking into account the following factors:

- Will the user try to achieve the right effect?
- Will the user notice that the correct action is available?
- Will the user associate the correct action with the effect they are trying to achieve?
- If the correct action is performed, will the user see that progress is being made toward solution of their task?

The two primary considerations when simulating the user's perspective were the currently completed lesson content and the DREAMM interface for the lesson. The DREAMM lesson modes detailed in Appendix .4.1 were constructed using the cognitive walkthrough process to ensure DREAMM functionality was consistent with the user's knowledge and expectations of the GUI in a given lesson.

Structured Interview

After each lesson retrospective probing [19] was performed in the form of a structured interview. The goal of the structured interview was to identify poorly addressed language concepts and confusing or inadequate material which would lower scalability, understandability, and transferability. Figure 9.8 shows the high level structure of the interview. During the user lessons, an audio recording was made to capture any "thinking out loud" and to record the structured interview. In addition, the computer screen was recorded to capture GUI interaction and mouse pointer identification of specific elements of SPNs and slides. The interview, audio, and video were all used to evaluate how well the training material met its goals and identify areas for improvement.

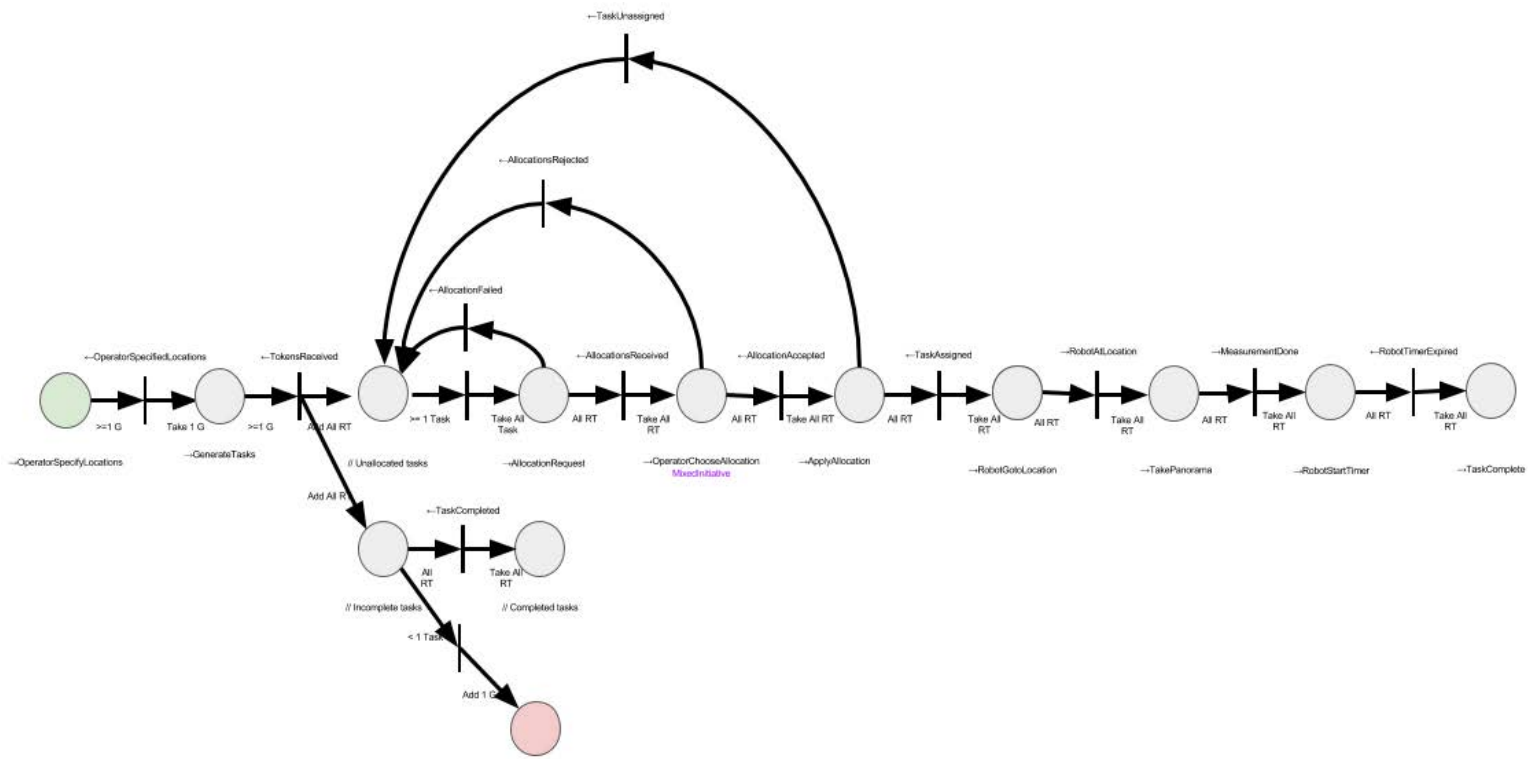


Figure 9.6: Asynchronous panorama SPN

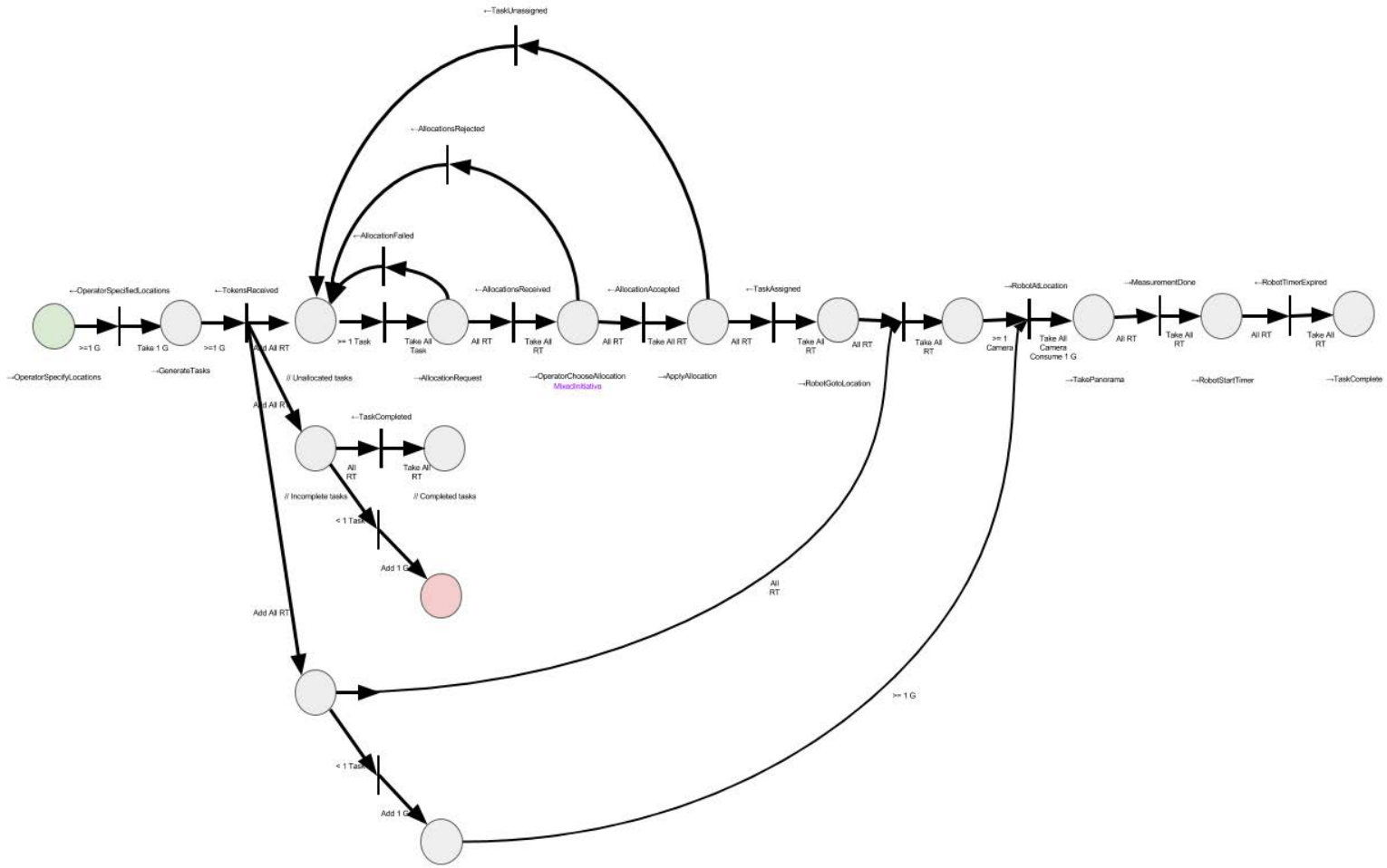


Figure 9.7: Synchronous panorama SPN

1. Review quizzes
   (a) Were you confident when answering this quiz?
   (b) If answered incorrectly, did you understand the correct answer?
2. Review jobs
   (a) Were you confident when performing this job?
   (b) If performed incorrectly, did you understand the explanation?
3. Amount of content
   (a) Were there any concepts you felt would benefit from additional slides?
   (b) Were there any concepts you felt had too many slides?
4. Feedback on slides
   (a) Were there any slides that caused confusion?
5. Feedback on DREAMM videos
   (a) Were there actions in DREAMM videos you were unable to recreate?
   (b) Were there actions in DREAMM you feel there should be a video for?
6. Feedback on DREAMM
   (a) Were there any actions in DREAMM you encountered difficulty performing?
7. Other sources of confusion
   (a) Were there any other sources of confusion in this lesson?

Figure 9.8: Retroactive probing dialogue

9.2 Study Results

Participant Demographics

In order to perform a time intensive and in-depth study, a study size of 6 participants was selected. Below are the recruitment demographics of the five non-experts and one novice. The novice user's results were used to begin identifying concepts and details which would need to be added to the lessons to broaden the accessible audience.

Gender:
5 / 6 Male (Novice)
1 / 6 Female

Last received diploma:
4 / 6 B.S. Engineering (Novice)
1 / 6 BS CS
1 / 6 HS diploma

Current program:
1 / 6 BS Humanities and Social Sciences
3 / 6 MS Engineering (Novice)
1 / 6 PhD CS
1 / 6 None

Which best describes your programming background? (select one)
1 / 6 No programming experience (Novice)
4 / 6 Do not have a CS degree, but have taken a class involving programming
1 / 6 Have a CS degree

Which best describes how often you typically program? (select one)
1 / 6 Monthly
3 / 6 Weekly
1 / 6 Daily
1 / 6 No programming experience (Novice)

List all programming languages you are comfortable programming in:
1 / 6 None (Novice)
1 / 6 Python, JavaScript
1 / 6 C++, Java, Python, MATLAB
1 / 6 C, C++, Matlab, R

1 / 6 MATLAB, Java
1 / 6 C

Lesson Durations

The *lesson durations* for each participant are shown in Table 9.1. This is the length of time from when they began the lesson to when the post-lesson interview began. Some lessons were designed to be longer than others based on the amount of content to be covered, number of quizzes, and number of jobs. Users could ask for assistance during the lesson if lesson content was unclear or unexpected behavior in DREAMM was encountered. Blue, bolded text indicates the participant completed that lesson slower than the average participant and red, unbolded text indicates the user completed that lesson quicker than average. User 5 is the novice user, and did not complete all lessons before the allocated time for the experiment expired.

User	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	Total
1	**2:50**	**28:24**	9:16	**8:25**	44:45	28:07	26:38	**54:46**	**1:07:05**	5:35	4:35:51
2	1:19	16:11	7:41	7:26	20:38	19:37	16:06	24:09	39:50	4:05	2:37:02
3	1:05	22:38	15:06	2:12	25:00	22:28	27:34	30:46	38:22	5:39	3:10:50
4	1:33	23:22	**51:56**	**21:08**	**1:08:37**	**1:05:26**	**56:03**	**53:58**	**1:28:44**	**9:00**	**7:19:47**
5	**2:37**	**37:13**	**40:37**	5:48	**1:38:34**	**34:52**	**55:38**	**1:12:57**	**1:07:29**	-	**6:55:45**
6	**2:10**	**27:48**	19:32	3:30	50:06	**37:48**	**38:41**	**54:48**	**1:12:26**	**9:33**	**5:16:22**
Avg	1:56	25:56	24:01	8:05	51:17	34:43	36:47	48:34	1:02:19	6:46	5:00:24

Table 9.1: Completion time for each lesson and each participant (hour: minutes: seconds) Bolded, blue durations indicate above average; red durations indicate below average

Lesson Scheduling

Table 9.2 shows the number of days between each lesson for each participant. Most sessions were scheduled within 2 days of the previous lesson, although there are some larger gaps due to the participant's travel schedule.

User	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	Final
1	-	0	2	0	**3**	0	0	**9**	2	**5**	0
2	-	0	2	0	**12**	2	1	2	2	2	0
3	-	0	**18**	0	0	0	1	0	0	1	0
4	-	0	1	0	0	1	**3**	0	1	0	0
5	-	0	**26**	0	0	0	2	2	0	1	0
6	-	0	0	0	1	1	1	0	**7**	**4**	0

Table 9.2: Number of days elapsed since previous lesson: Bolded, red quantities are greater than 2 days

Quiz Results

Table 9.3 shows the number of the quizzes answered correctly for each user and lesson. In general, the total number of quizzes correlates with the length and complexity of the lesson. Quizzes were presented either directly after presentation of a new concept to evaluate understanding or at the beginning of a lesson to review earlier concepts that would be used frequently in the lesson.

User	L2	L3	L4	L5	L6	L7	L8	L9
1	8/8	4/4	2/2	**28/29**	12/12	4/4	3/3	4/4
2	8/8	4/4	2/2	29/29	**11/12**	4/4	3/3	4/4
3	8/8	4/4	2/2	29/29	12/12	4/4	3/3	4/4
4	8/8	4/4	2/2	29/29	12/12	4/4	3/3	4/4
5	8/8	4/4	2/2	29/29	12/12	**2/4**	3/3	4/4
6	8/8	4/4	2/2	29/29	12/12	4/4	3/3	4/4

Table 9.3: Number of quizzes answered correctly for each lesson and each participant: Bolded, red numbers indicate at least one answer was incorrect.

Job Results

User	L2	L3	L5	L6	L7	L8	L9	L10
1	2/2	2/2	2/2	1/1	**2/4**	**6/6**	**1/2**	2/2
2	2/2	2/2	2/2	1/1	4/4	6/6	**1/2**	2/2
3	2/2	2/2	2/2	1/1	**3/4**	6/6	2/2	2/2
4	2/2	2/2	2/2	1/1	**2/4**	6/6	2/2	2/2
5	2/2	2/2	2/2	1/1	**0/4**	**3/6**	**1/2**	-
6	2/2	2/2	2/2	1/1	**3/4**	4/4	**1/2**	2/2

Table 9.4: Number of jobs performed correctly for each lesson and each participant: Red numbers indicate at least one job was not performed correctly

Final Plan

All five non-experts constructed solutions for both versions of the final plan. The beginner user did not begin the final plan due to time constraints.

For the asynchronous version of the plan, 3 of the 5 participants were able to construct a correct solution. For the synchronous version of the plan, none of the participants were able to construct a correct solution.

9.3 Analysis

9.3.1 Demographic

The novice participant experienced many difficulties with the lessons, and did not begin Lesson 10 nor the Final Plan as training time had expired. With one exception, the completed lessons were performed in an above average amount of time. In addition, the participant requested clarification during lessons on concepts such as as variables and looping.

On average, the non-experts completed the ten lessons in 4 hours and 35 minutes. User feedback largely consisted of suggestions to improve clarity of instruction, quizzes, and jobs, with only a few instances of requests for additional content. These requests are detailed below in Section 9.3.3 and Section 9.3.4. Overall, the feedback suggests the lessons were scoped appropriately for users with exposure to programming and computer science.

9.3.2 Language Scalability

To improve scalability, training material should not require an expert user's presence. The expert user present during the study performed several tasks: quiz grading, job grading, task clarification, and reviewing earlier concepts. The answer format for the quizzes can

be sorted into 4 categories: free response, multiple choice, labeling a provided plan, and adding to a provided plan. To automate free response quiz grading, the questions would converted to multiple choice answers. Users' free response answers and retroactive probing dialog would be used to select a set of answers. Multiple choice answers can easily be graded programmatically. Quizzes involving labeling a plan, i.e. updating where tokens will be, have exactly one correct answer and can be graded automatically. Quizzes involving adding to a provided plan are similar to jobs, but were performed on a paper handout instead of DREAMM. These would be converted to jobs.

Automatic grading of jobs is possible by comparing the user's plan file to an "answer key" plan file. Certain flexibility would be necessary, such as allowing multiple events on a vertex to be placed in any order and comparing variable mapping and not specific variable names.

Task clarification was necessary in several lessons. Updating the training material accordingly to eliminate these sources of confusion may be sufficient to remove the need for an expert for task clarification. Otherwise, additional user studies may be necessary to iteratively improve lesson clarity.

One of the most common pieces of feedback was the desire for a mechanism to review topics on demand during lessons. Several concepts were proposed, including adding a glossary of terms with links to relevant lessons. A second option was adding tool text to keyword concepts, with the hover text providing a brief refresher and links to relevant lessons. A third option was to create a wiki of key terms, with hyperlink encyclopedia references to other key terms used in an entry. Given all three of these concepts use similar content, a future study could compare which combination of approaches is most useful for users.

9.3.3 Language Learning

Quizzes were used as the primary assessment of a user's understanding of language syntax. The results in Table 9.3 show that incorrect quiz answers were uncommon: in both instances where a non-expert user answered a quiz incorrectly, they indicated in the structured interview that the content was understood but that the quiz's wording was confusing. None of the quizzes reviewing concepts from previous lessons were answered incorrectly. We hypothesize this was due to the generally short time gap between most lessons for participants, as shown in Table 9.2.

The beginner user had difficulty answering a quiz question involving looping. It would be necessary to add additional content explaining looping to address this difficulty.

A common request during post-lesson interviews was more detailed explanations of various robotics and programming terms used in the lessons, including *system*, *runtime*, and *variable field*. Explanations relating the terms to the Cooperative Robotic Watercraft project were understood and would be a useful addition to the training material.

9.3.4 Language Application

Job Results

Jobs results were one method used to assess a user's ability to transfer knowledge of the language into usable team plans. Table 9.4 shows that each user had difficulty with at least one job.

In Lesson 7 a common source of confusion was using "Relevant Token" edge requirements for edges connected to transitions with no input events. Additional content and a quiz could be used to reinforce that "Robot" token edge requirements must be used for these transitions. Another source of confusion was adding looping to plans based on a question presented to the operator. Figure 9.9a shows a mistake where looping moved tokens too far back in the plan, where other edge requirements would prevent the the moved token from creating triggering the behavior additional times. A quiz reminding developers to consider other existing edge requirements could be effective in addressing this type of semantic error.

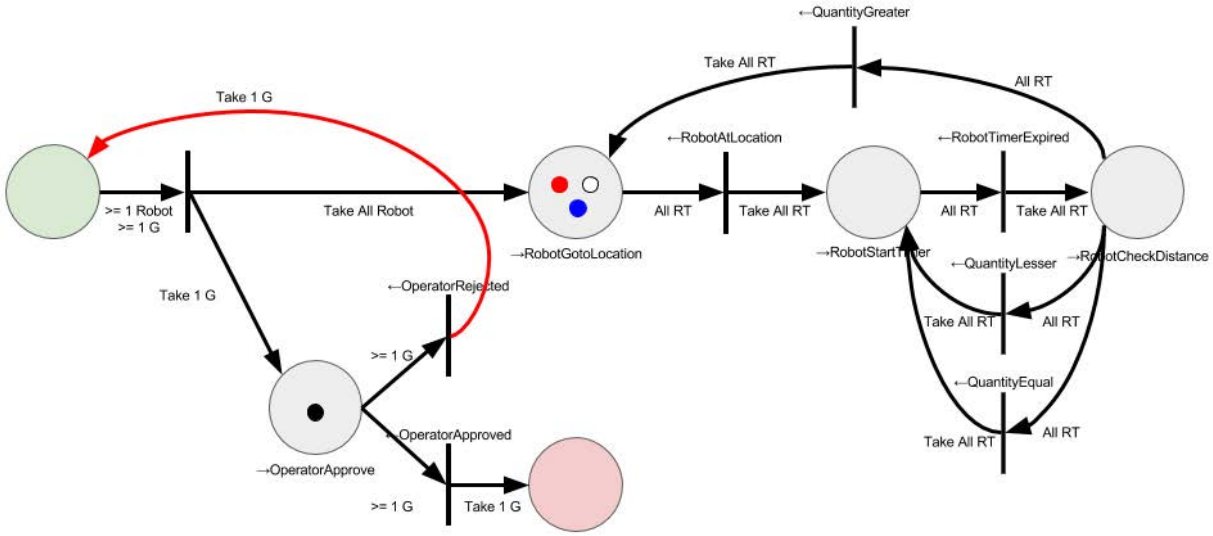
Several users were confused in Lesson 8 by how to use the map widget to provide static definitions as the tutorial video showed how to provide a static definition for a string using a text field. Additional video content demonstrating map usage would address this confusion.

In Lesson 9 several users were confused by the instructions for a job designed to test their ability to use variables correctly. One source of confusion was the names to use for variables. Examples in the lesson assigned variable names to fields for certain events, but not all of the events used in the job. Some participants did not know to choose their own names to assign to variables for these other event's fields. In addition, the job of assigning write and read variable names to all input and output event's fields for an entire plan was found to be overwhelming. These participants suggested adding an additional job earlier with a similar task but for a smaller plan. A second source of errors involved the @task variable. The @task variable is specified as a read variable for output event fields to indicate that when a task token activates the event, that field's value should be read from the corresponding task. For example, in the final plans (Figure 9.6 and Figure 9.7) robots are assigned tasks which are tied to a particular location. The @task variable is used as the read variable for the Location field in the Robot Goto Location output event so that when a task token enters the place, the robot assigned to that task is instructed to goto the task's location. Some participants referenced the @task variable incorrectly, such as listing it as the read variable for the Generate Task output event's TaskClass field instead of providing a definition such as Temperature or Camera.

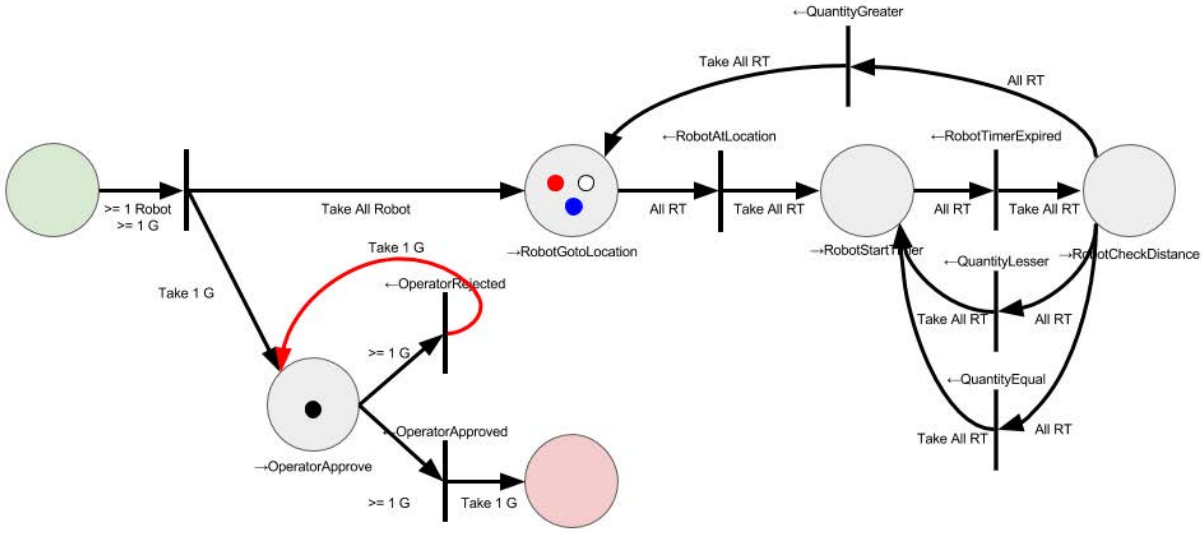
Final Plan Results

The two versions of the final plan were another method used to gauge participants' abilities to apply the training material.

Due to time constraints, the beginner user did not have time to begin the final plan.



(a) Incorrect implementation of looping blocked by >= 1 Robot requirement



(b) Correct implementation of looping

Figure 9.9: Confusion about looping

All non-expert users produced a plan to address the initial, asynchronous version of the plan. Figure 9.6 shows a plan meeting all of the specified requirements. Three participants made plans identical to this, with inconsequential differences such as variable names. The other two participants made errors when addressing the requirement "When the task tokens are received, they should be allocated immediately by the System AI." The desired method was to use the "Mixed Initiative" markup on "→Operator Select Allocation" output event, and using the "Immediate" option to indicate the autonomy should immediately make the decision for the operator. An earlier example showed how to use "Mixed Initiative" markup to timeout an operator decision after 10 seconds. Instead, several users removed the place with the "→Operator Select Allocation" output event and the transitions with the "←AllocationsRejected" and "←AllocationAccepted" input events. This connected the transition with the "←AllocationsReceived" input event to the place with the "→ApplyAllocation" output event. However, as "←AllocationsReceived" returns a list of allocations and "→ApplyAllocation" requires a single allocation, their variables are incompatible, leading to the "→ApplyAllocation" being undefined. Participants indicated naming an event "Operator Select Allocation" implied an operator would always be involved. Renaming the event or providing an example of this use of the Mixed Initiative markup to the lesson could address this confusion.

None of the users were able to complete the synchronous version of the plan without assistance. A correct solution to this version of the requirements is shown in Figure 9.7. Participants were unable to encode detection of when all boats were at their task's location.

Lesson 9 introduced a pattern to determine when all tasks were finished, which was used to transition the plan to an end place, seen in Figure 9.5. A similar pattern using "Robot At Location" instead of "Task Completed" could be used to determine when all task's robots were in position, but the lesson did not cover variations of the pattern. This variation of the final plan was designed to gauge if additional content would be necessary for users to recognize and repurpose the "all tasks finished" pattern. One participant identified that the all tasks finished would be used, but could not determine the exact format. Another participant proposed methods for synchronizing the action which would require unsupported events and/or functionality in the language, such as adding an input event which would be generated when all paths were finished, or using a variable on an in-edge requirement to set the required number of robot tokens on an in edge to the number of tasks. In the post-lesson interview, participants recommended adding additional lesson content specifically addressing repurposing the all tasks finished structure.

9.3.5 Summary

In this chapter, we discussed a process used to design training material to teach non-experts how to design SPNs and a model for evaluating the effectiveness of the material. Cognitive walkthrough, incremental complexity, and a pilot study were used to design the material,

while retrospective probing and structured interviews were used to asses learnability. Designing a small set of plans with incremental complexity required significant development effort and several iterations as scope of the lesson material was increased and reduced based on total duration. However, my intuition is that it is of key importance to quickly training users.

To support our first contribution of a team planning language for non-experts, we conducted a user study with 5 non-expert programmers and one novice programmer. Results for the target demographic of non-experts were encouraging, with an average completion time of 5 hours. We choose to recruit 5 non-experts based on several usability study recommendations and the required time involvement, which is a small number compared to other types of user studies. Consistent with other usability studies, I found that the users for the large part were confused and made mistakes in similar areas. If resources made a 10 user study possible, I believe it would be more beneficial to run two studies of 5 users, improving material in between, than to conduct a single study of 10 users.

Chapter 10

Discussion

10.1 Summary

This thesis document has presented research efforts and evidence of 4 contributions.

SPN Team Plan Language We designed the SAMI Petri Net language as an extension of Colored Petri Nets. Extensions were designed to add compact and simple representation power of complex concepts such as team coordination, contingency invocation, and task assignment. These extensions were then analyzed to find reduction rules to allow for the use of existing Petri Net analysis techniques in plan development assistant agents. Additional agents were developed using meta-knowledge of events in the domain to provide additional tools to developers. We presented SPN plans designed in two domains, one using autonomous watercraft and one using autonomous aerial vehicles, to demonstrate the ability of the language to capture plans in multiple domains. Plans were executed and refined in the field for a team of autonomous boats to find useful extensions to the language, such as operator interrupts, and to test the limits of the language. A simulation environment was then used to evaluate the impact of operator interrupt mechanism and results showed that both hazard resolution time and operator involvement decreased as desired. A set of lessons to teach non-experts to develop team plans was designed and evaluated in a user study, which showed that that non-experts were able to successfully design team plans after an average lesson completion time of 5 hours.

SAMI Markup Language We designed the SAMI (Situational Awareness and Mixed Initiative) markup language for use in human-robot team plans. We applied markup to the SPNs we developed in two domains and designed operator UIs for each to demonstrate its flexibility across domains. Using lessons from extensive field deployments, the markup vocabulary was expanded and plan markup was modified to better fulfill the needs of the team. We also presented a simulation designed to evaluate how well markup could affect

team behavior at specific states in a team plan. The results of the experiment showed that the markup language was able to prioritize boat safety over data collection only when a boat was in physical danger. Finally, our user study showed that the markup concept was understood by non-expert users, but that naming conventions for markup keywords should be carefully considered to prevent confusion.

Field Experience For our third contribution we presented experience gained through extensive field deployments. We described design tradeoff decisions made over several years of hardware iterations with the goal of improving capability of a large robot team deployed by a small number of operators. We described operational hazards at several deployment sites and how team plan and human-robot interaction was adjusted to operate within those conditions. We found wireless connectivity issues due to water absorption to be the primary and unanticipated source of failure for the team, but were able to design and test team plans to assist in the recognition and recovery of robots in the event of connectivity failure. To address this failure mode, hardware upgrades supporting an external antenna would be required, though software framework upgrades to support decentralized, ad-hoc networks could also be a solution.

Training Design and Evaluation In our fourth contribution we demonstrated a method for creating training material for a team planning language. We presented a strategy using a combination of cognitive walkthrough, incremental layering of language complexity to existing material, and a pilot study to create a set of 10 lessons which could be converted entirely to online media. In addition, we presented techniques for conducting a learnability study designed to evaluate the effectiveness of developed material. The lessons incorporate quizzes and jobs to evaluate a user's understanding of language syntax and ability to apply it. Our learnability user study also used retrospective probing and structured interview techniques to identify shortcomings in the material which can be addressed and iterated upon.

These 4 contributions provide a powerful system for human-robot teams which are ready to be put in the hands of users. Using this framework and training material, non-experts interested in using multi-robot teams to achieve tasks have a lower barrier to entry. In addition, our framework allows for several desirable improvements for collaboration between domain experts, human factors experts, and planning experts. The presented architecture separates team plans, UI components, and service algorithms, reducing complications due to collaboration across a shared codebase. This separation, combined with our domain configuration file architecture, also allows for UI components, service algorithms, and plans to easily be shared between collaborators, allowing for greater impact of research efforts.

10.2 Future Topics

10.2.1 Geographic Markup

One useful extension to the SAMI markup language would be geographic markup, which would be attached to regions on a map instead of specific events in the plan. In the plans presented in Chapter 6, markup related to geographic factors worked well as dangerous regions were typically associated with specific sections of the plan, such as moving from the shore to the open water area. When this is not the case, it would be useful to modify priorities, optimization factors, and levels of autonomy based on if the associated team member is inside or outside of specific regions. One of the challenges in this concept is how to resolve scenarios where the same category of markup should be applied to an event both as geographic markup and conventional event markup.

10.2.2 Designer Development

Another area of future research is enabling users without programming exposure to use the language. The results of the user study in Chapter 9 suggest more time and training material would be necessary for a user without programming exposure to write their own plans. Another strategy is to simplify writing plans, sacrificing representational power to gain accessibility. One method currently being investigated which adopts the simplification strategy is a new *Designer* mode in the DREAMM IDE presented in Chapter 5, which is similar in concept to the MissionLab wizard [57].

In this mode a *designer* navigates through a dialog tree of questions about the desired plan's behavior and upon completion the system generates a corresponding SPN. Figure 10.1 shows an example of this dialog tree where the designer has requested an exploration plan prioritizing energy efficiency. Grey, dashed boxes are *question nodes* which represent some decision requiring the designer's input. Each question node has a number of children called *option nodes*, represented as solid boxes. Green boxes indicate the option node selected by the designer, leaving the unselected options red. For visual clarity, only two of the question nodes have their option nodes visible. Decisions near the root of the tree would determine high level characteristics of the generated plan, determining which SPN template to use, and decisions near the leafs would determine low level details, such as markup on events. Hand-coding a SPN for each possible enumeration of the decision tree would be time consuming. To address this, an underlying system of SPN "fragments" is developed by an SPN expert which correspond to option nodes. When the SPN expert creates the decision tree which is presented to the designer, the expert also creates rules describing how to combine fragments when their corresponding option nodes are selected.

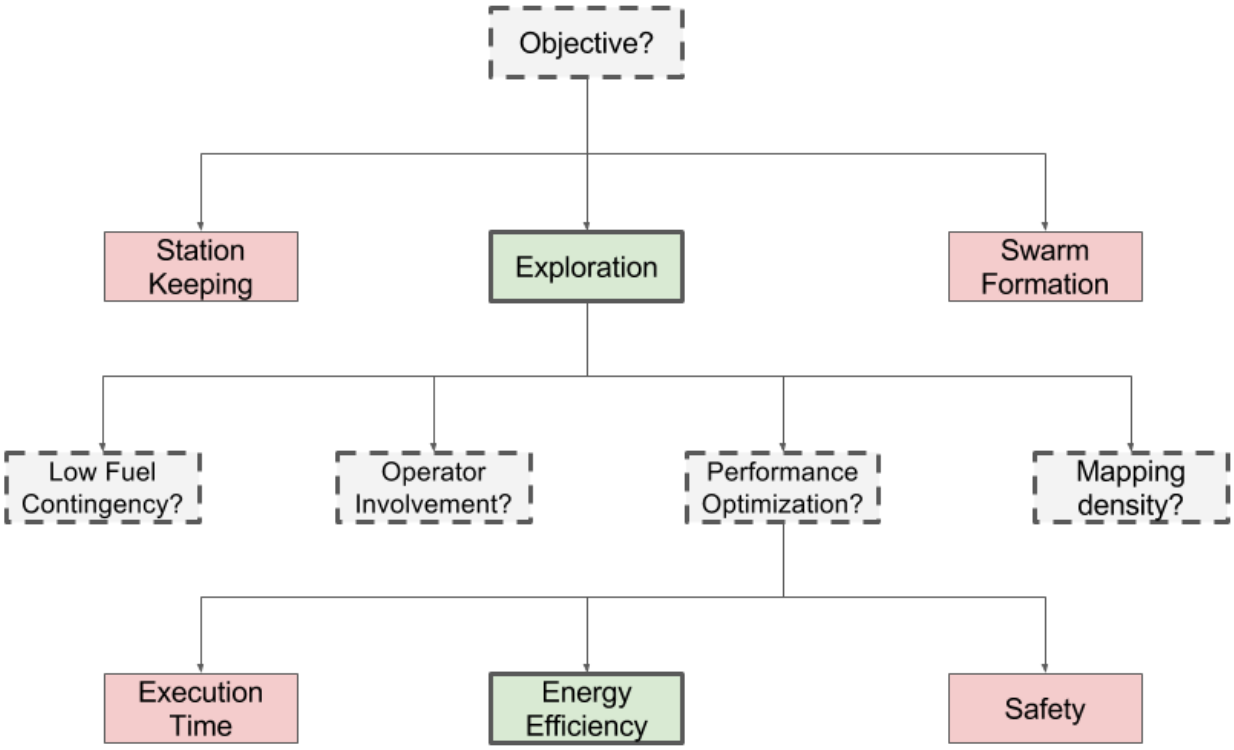


Figure 10.1: Sample tree for Designer workflow

10.2.3 Markup Transferability Across Devices

SAMI markup was designed to provide a sufficient level of information for a UI to act on without overfitting for a particular domain or UI style. By designing SPNs in multiple domains we can identify shortcomings in the markup language's transferability across domains. One method to evaluate how well markup translates across different UI styles would be to design a SAMI compatible UI for new types of control devices. In Chapter 5, we focused on a GUI designed for a laptop selected for its balance of portability, display size, and ability to compile code. In future research, shortcomings in markup interpretation could be analyzed when using the same SPN on a more portable device, such as a tablet or smartphone. These devices allow the operator to easily move around the test site, but have a smaller screen size and are typically constrained to touch screen interaction. We anticipate the greater mobility of the devices improves the operator's situational awareness, but increases the importance of markup's ability to capture interaction priority and mixed initiative autonomy as user interactions become less frequent due to the limited visualization space and slower interaction speed. In the other extreme, plans could be tested on a remote, multi-display workstation. In this case, we anticipate interaction priority and mixed-initiative autonomy markup to be less important as visualization space and interaction speed is improved, but situational awareness markup to become critical.

10.2.4 Decentralized Execution

In the presented work we have assumed a centralized communication model due to the high level of involvement of an operator. However, in some domains a distributed model may be preferred for robustness and expandability [54]. To support decentralized execution, future research would develop algorithms similar to those in PNP [201] to take a centralized SPN and generate distributed CPNs with appropriate synchronization points. One challenge in developing this type of algorithm is the ability to dynamically create tasks and subteams during an SPN's execution. This would require dynamically determining the team-members involved at each synchronization point. In practice, a decentralized plan may require more than these synchronization points. Contingencies for identifying and addressing communication failure may differ for a centralized model versus a decentralized model.

Bibliography

[1] Shafiq Abedin, Michael Lewis, Nathan Brooks, Sean Owens, Paul Scerri, and Katia Sycara. Suave. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 55(1):91–94, 2011.

[2] Julie A Adams. Human-robot interaction design: Understanding user needs and requirements. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 447–451. SAGE Publications Sage CA: Los Angeles, CA, 2005.

[3] Julie A Adams. Unmanned vehicle situation awareness: A path forward. In *Human systems integration symposium*, pages 31–89. Citeseer, 2007.

[4] Julie A Adams. Multiple robot/single human interaction: effects on perceived workload. *Behaviour & Information Technology*, 28(2):183–198, 2009.

[5] Julie A Adams, Ruzena Bajcsy, Jana Kosecka, V Kumar, Robert Mandelbaum, Max Mintz, R Paul, Curtis Wang, Yoshio Yamamoto, and Xiaoping Yun. Cooperative material handling by human and robotic agents: Module development and system synthesis. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 200–205. IEEE, 1995.

[6] Rajeev Alur. Formal analysis of hierarchical state machines. In *Verification: Theory and Practice*, pages 42–66. Springer, 2003.

[7] Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for petri nets. *Theoretical Computer Science*, 3(1):85–104, 1976.

[8] Ronald C Arkin, Tucker Balch, et al. Cooperative multiagent robotic systems. *Artificial Intelligence and Mobile Robots. MIT/AAAI Press, Cambridge, MA*, 1998.

[9] Ronald C Arkin, Yoichiro Endo, and Douglas Christopher MacKenzie. Usability evaluation of high-level user assistance for robot mission specification. 2002.

[10] Marcelo Gabriel Armentano and Analía Amandi. Plan recognition for interface agents. *The Artificial Intelligence Review*, 28(2):131, 2007.

[11] Quan Bai, Minjie Zhang, and Khin Than Win. A colored petri net based approach for multi-agent interactions. In *Proc. of 2nd International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand*, pages 152–157, 2004.

[12] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and João Valente. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *Journal of Field Robotics*, 28(5):667–689, 2011.

[13] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. Optimizing schedules for prioritized path planning of multi-robot systems. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 271–276. IEEE, 2001.

[14] Markus Bernard, Konstantin Kondak, Ivan Maza, and Anibal Ollero. Autonomous transportation and deployment with aerial robots for search and rescue missions. *Journal of Field Robotics*, 28(6):914–931, 2011.

[15] Luca Bernardinello and Fiorella De Cindio. A survey of basic net models and modular net classes. In *Advances in Petri Nets 1992*, pages 304–351. Springer, 1992.

[16] Graeme Best, Jan Faigl, and Robert Fitch. Multi-robot path planning for budgeted active perception with self-organising maps. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3164–3171. IEEE, 2016.

[17] Bradley J Betts, Robert W Mah, Richard Papasin, Rommel Del Mundo, Dawn M McIntosh, and Charles Jorgensen. Improving situational awareness for first responders via mobile computing. 2005.

[18] Deepak Bhadauria, Onur Tekdas, and Volkan Isler. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics*, 28(3):388–404, 2011.

[19] Julie H Birns, Kristen A Joffre, Jonathan F Leclerc, and Christine Andrews Paulsen. Getting the whole picture: Collecting usability data using two methods—-concurrent think aloud and retrospective probing. In *Proceedings of UPA Conference*, pages 8–12. Citeseer, 2002.

[20] Bert Bos, Håkon Wium Lie, Chris Lilley, and Ian Jacobs. Cascading style sheets, level 2 css2 specification, 1998.

[21] Sylvia C Botelho and Rachid Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1234–1239. IEEE, 1999.

[22] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 16:16, 1998.

[23] Frank Broz, Illah Nourbakhsh, and Reid Simmons. Planning for human–robot interaction in socially situated tasks. *International Journal of Social Robotics*, 5(2):193–214, 2013.

[24] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2383–2388. IEEE, 2002.

[25] Andrea Bunt, Cristina Conati, and Joanna McGrenere. Supporting interface customization using a mixed-initiative approach. In *Proceedings of the 12th international conference on Intelligent user interfaces*, pages 92–101. ACM, 2007.

[26] Jennifer L Burke, Robin R Murphy, Michael D Coovert, and Dawn L Riddle. Moonlight in miami: Field study of human-robot interaction in the context of an urban search and rescue disaster response training exercise. *Human–Computer Interaction*, 19(1-2):85–116, 2004.

[27] Nadia Busi. Analysis issues in petri nets with inhibitor arcs. *Theoretical Computer Science*, 275(1-2):127–177, 2002.

[28] Maria Paola Cabasino, Alessandro Giua, Stéphane Lafortune, and Carla Seatzu. Diagnosability analysis of unbounded petri nets. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 1267–1272. IEEE, 2009.

[29] Linqin Cai, Tao Mei, Yining Sun, Lei Sun, and Zuchang Ma. Modeling and analyzing multi-agent task plans for intelligent virtual training system using petri nets. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 1, pages 4766–4770. IEEE, 2006.

[30] Nannan Cao, Kian Hsiang Low, and John M Dolan. Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 7–14. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[31] Jessie YC Chen, Michael J Barnes, and Michelle Harper-Sciarini. Supervisory control of multiple robots: Human-performance issues and user-interface design. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(4):435–454, 2011.

[32] Shih-Yi Chien, Michael Lewis, Siddharth Mehrotra, Shuguang Han, Nathan Brooks, Huadong Wang, and Katia Sycara. Task switching for supervisory control of multi-robot teams. *IEEE Transactions on Human-Machine Systems*, 2016.

[33] Søren Christensen and Niels Hansen. Coloured petri nets extended with place capacities, test arcs and inhibitor arcs. *Application and Theory of Petri Nets 1993*, pages 186–205, 1993.

[34] Timothy H Chung, Michael R Clement, Michael A Day, Kevin D Jones, Duane Davis, and Marianna Jones. Live-fly, large-scale field experimentation for large numbers of fixed-wing

uavs. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1255–1262. IEEE, 2016.

[35] Andrew S Clare, Pierre CP Maere, and Mary L Cummings. Assessing operator strategies for real-time replanning of multiple unmanned vehicles. *Intelligent Decision Technologies*, 6(3):221–231, 2012.

[36] Edward Clarkson and Ronald C Arkin. Applying heuristic evaluation to human-robot interaction systems. In *Flairs Conference*, pages 44–49, 2007.

[37] Peter Corke, Carrick Detweiler, Matthew Dunbabin, Michael Hamilton, Daniela Rus, and Iuliu Vasilescu. Experiments with underwater robot localization and tracking. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4556–4561. IEEE, 2007.

[38] Peter Corke, Stefan Hrabar, Ron Peterson, Daniela Rus, Srikanth Saripalli, and Gaurav Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3602–3608. IEEE, 2004.

[39] Hugo Costelha and Pedro Lima. Modelling, analysis and execution of multi-robot tasks using petri nets. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1187–1190. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[40] Hugo Costelha and Pedro Lima. Robot task plan representation by petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, 33(4):337–360, 2012.

[41] Carle Côté, Dominic Létourneau, François Michaud, J-M Valin, Yannick Brosseau, Clement Raievsky, Mathieu Lemay, and Victor Tran. Code reusability tools for programming mobile robots. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 1820–1825. IEEE, 2004.

[42] Jennifer Cross, Christopher Bartley, Emily Hamner, and Illah Nourbakhsh. A visual robot-programming environment for multidisciplinary education. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 445–452. IEEE, 2013.

[43] Jennifer L Cross, Emily Hamner, Chris Bartley, and Illah Nourbakhsh. Arts & bots: application and outcomes of a secondary school robotics program. In *Frontiers in Education Conference (FIE), 2015 IEEE*, pages 1–9. IEEE, 2015.

[44] Mary L Cummings and Paul J Mitchell. Predicting controller capacity in supervisory control of multiple uavs. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(2):451–460, 2008.

[45] Torbjørn S Dahl, Maja J Mataric, and Gaurav S Sukhatme. Multi-robot task-allocation through vacancy chains. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 2293–2298. IEEE, 2003.

[46] Philippe Darondeau, Stephane Demri, Roland Meyer, and Christophe Morvan. Petri net reachability graphs: Decidability status of first order properties. *arXiv preprint arXiv:1210.2972*, 2012.

[47] Jnaneshwar Das, Thom Maughan, Mike McCann, Mike Godin, Tom O'Reilly, Monique Messié, Fred Bahr, Kevin Gomes, Frédéric Py, James G Bellingham, et al. Towards mixed-initiative, multi-robot field experiments: Design, deployment, and lessons learned. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3132–3139. IEEE, 2011.

[48] Chandan Datta, Chandimal Jayawardena, I Han Kuo, and Bruce A MacDonald. Robostudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2352–2357. IEEE, 2012.

[49] Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg. *Lectures on concurrency and Petri nets: advances in Petri nets*, volume 3098. Springer, 2004.

[50] James P Diprose, Bruce A MacDonald, and John G Hosking. Ruru: A spatial and interactive visual programming language for novice robot programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 25–32. IEEE, 2011.

[51] Stephen R Dixon, Christopher D Wickens, and Dervon Chang. Mission control of multiple unmanned aerial vehicles: A workload analysis. *Human factors*, 47(3):479–487, 2005.

[52] Miguel Duarte, Vasco Costa, Jorge Gomes, Tiago Rodrigues, Fernando Silva, Sancho Moura Oliveira, and Anders Lyhne Christensen. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PloS one*, 11(3):e0151834, 2016.

[53] Catherine Dufourd, Alain Finkel, and Ph Schnoebelen. Reset nets between decidability and undecidability. *Automata, Languages and Programming*, pages 103–115, 1998.

[54] James Edmondson and Douglas Schmidt. Multi-agent distributed adaptive resource allocation (madara). *International Journal of Communication Networks and Distributed Systems*, 5(3):229–245, 2010.

[55] Tarek El-Gaaly, Christopher Tomaszewski, Abhinav Valada, Prasanna Velagapudi, Balajee Kannan, and Paul Scerri. Visual obstacle avoidance for autonomous watercraft using smartphones. 2013.

[56] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, 2012.

[57] Yoichiro Endo, Douglas C MacKenzie, and Ronald C Arkin. Usability evaluation of high-level user assistance for robot mission specification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(2):168–180, 2004.

[58] Alessandro Farinelli, Masoume M Raeissi, Nathan Brooks, Paul Scerri, et al. Interacting with team oriented plans in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, pages 1–30, 2016.

[59] Mark Fiala. A robot control and augmented reality interface for multiple robots. In *Computer and Robot Vision, 2009. CRV'09. Canadian Conference on*, pages 31–36. IEEE, 2009.

[60] Leah Findlater and Joanna McGrenere. Beyond performance: Feature awareness in personalized interfaces. *International Journal of Human-Computer Studies*, 68(3):121–137, 2010.

[61] Alain Finkel, Gilles Geeraerts, J-F Raskin, and Laurent Van Begin. On the ω-language expressive power of extended petri nets. *Electronic Notes in Theoretical Computer Science*, 128(2):87–101, 2005.

[62] Gerhard Fischer. User modeling in human–computer interaction. *User modeling and user-adapted interaction*, 11(1-2):65–86, 2001.

[63] Robert Fisher, Thomas Kollar, and Reid Simmons. Building and learning from a contextual knowledge base for a personalized physical therapy coach. In *ICML Workshop on Robot Learning, Atlanta GA*, volume 3, 2013.

[64] Jerry L Franke, Vera Zaychik, Thomas M Spura, and Erin E Alves. Inverting the operator/vehicle ratio: Approaches to next generation uav command and control. *Proceedings of AUVSI Unmanned Systems North America*, 2005, 2005.

[65] Harry Funk, Robert Goldman, Christopher Miller, John Meisner, and Peggy Wu. A playbook tm for real-time, closed-loop control. 2006.

[66] Allison Funkhouser and Reid Simmons. Annotation of utterances for conversational nonverbal behaviors. In *International Conference on Social Robotics*, pages 521–530. Springer International Publishing, 2016.

[67] Krzysztof Z Gajos, Daniel S Weld, and Jacob O Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12):910–950, 2010.

[68] Fei Gao, Mary L Cummings, and Erin Treacy Solovey. Modeling teamwork in supervisory control of multiple robots. *IEEE Transactions on Human-Machine Systems*, 44(4):441–453, 2014.

[69] Fei Gao, M.L. Cummings, and L.F. Bertuccelli. Teamwork in controlling multiple robots. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE Intl. Conf. on*, pages 81–88, march 2012.

[70] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[71] Ashish Ghoda and Mamta Dalal. *XAML developer reference*. Pearson Education, 2011.

[72] M.A. Goodrich, D.R. Olsen, J.W. Crandall, and T.J. Palmer. Experiments in adjustable autonomy. In *Proc. of IJCAI Workshop on Autonomy, Delegation and Control: Interacting with Intelligent Agents*, pages 1624–1629, 2001.

[73] Michael A Goodrich, Joseph L Cooper, Julie A Adams, Curtis Humphrey, Ron Zeeman, and Brian G Buss. Using a mini-uav to support wilderness search and rescue: Practices for human-robot teaming. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6. IEEE, 2007.

[74] Michael A Goodrich, Timothy W McLain, Jeffrey D Anderson, Jisang Sun, and Jacob W Crandall. Managing autonomy in robot teams: observations from four experiments. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 25–32. ACM, 2007.

[75] Michael A Goodrich, Bryan S Morse, Cameron Engh, Joseph L Cooper, and Julie A Adams. Towards using unmanned aerial vehicles (uavs) in wilderness search and rescue: Lessons from field trials. *Interaction Studies*, 10(3):453–478, 2009.

[76] Michael A Goodrich, Bryan S Morse, Damon Gerhardt, Joseph L Cooper, Morgan Quigley, Julie A Adams, and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[77] Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: a cognitive dimensions framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[78] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 649–658. ACM, 2009.

[79] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[80] Sunao Hashimoto, Akihiko Ishida, Masahiko Inami, and Takeo Igarashi. Touchme: An augmented reality based remote robot manipulation. In *The 21st International Conference on Artificial Reality and Telexistence, Proceedings of ICAT2011*, 2011.

[81] Daniel Heß and C Rohrig. Remote controlling of technical systems using mobile devices. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009. IEEE International Workshop on*, pages 625–628. IEEE, 2009.

[82] Dirk Heylen, Stefan Kopp, Stacy C Marsella, Catherine Pelachaud, and Hannes Vilhjálmsson. The next step towards a function markup language. In *International Workshop on Intelligent Virtual Agents*, pages 270–280. Springer, 2008.

[83] Eli R Hooten. A mobile, map-based tasking interface for human-robot interaction. Technical report, DTIC Document, 2010.

[84] Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 305–313. Morgan Kaufmann Publishers Inc., 1999.

[85] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[86] Andrew Howard, Lynne E Parker, and Gaurav S Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research*, 25(5-6):431–447, 2006.

[87] M Ani Hsieh, Anthony Cowley, James F Keller, Luiz Chaimowicz, Ben Grocholsky, Vijay Kumar, Camillo J Taylor, Yoichiro Endo, Ronald C Arkin, Boyoon Jung, et al. Adaptive teams of autonomous aerial and ground robots for situational awareness. *Journal of Field Robotics*, 24(11-12):991–1014, 2007.

[88] Curtis M Humphrey and Julie A Adams. Compass visualizations for human-robotic interaction. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 49–56. ACM, 2008.

[89] Curtis M Humphrey, Christopher Henk, George Sewell, Brian W Williams, and Julie A Adams. Assessing the scalability of a multiple robot interface. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pages 239–246. IEEE, 2007.

[90] Anthony Jameson. Adaptive interfaces and agents. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, 105, 2009.

[91] Kurt Jensen. Coloured petri nets. *Petri nets: central models and their properties*, pages 248–299, 1987.

[92] Kurt Jensen and Lars M Kristensen. *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media, 2009.

[93] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213, 2007.

[94] Edward Gil Jones, Brett Browning, M Bernardine Dias, Brenna Argall, Manuela Veloso, and Anthony Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 570–575. IEEE, 2006.

[95] D. Kaber and M. Endsley. The effects of level of automation and adaptive automation on human performance, situation awareness and workload in a dynamic control task. *Theoretical Issues in Ergonomics Science*, 5(2):113–153, 2004.

[96] Gal A Kaminka and Inna Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 20, page 108. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[97] Gal A Kaminka and Inna Frenkel. Integration of coordination mechanisms in the bite multi-robot architecture. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2859–2866. IEEE, 2007.

[98] George Kantor, Sanjiv Singh, Ronald Peterson, Daniela Rus, Aveek Das, Vijay Kumar, Guilherme Pereira, and John Spletzer. Distributed search and rescue with robot and sensor teams. In *Field and Service Robotics*, pages 529–538. Springer, 2003.

[99] Erez Karpas, Steven James Levine, Peng Yu, and Brian Charles Williams. Robust execution of plans for human-robot teams. In *ICAPS*, pages 342–346, 2015.

[100] Jun Kato, Daisuke Sakamoto, Masahiko Inami, and Takeo Igarashi. Multi-touch interface for controlling multiple mobile robots. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, pages 3443–3448. ACM, 2009.

[101] Kazuhiko Kawamura, Phongchai Nilas, Kazuhiko Muguruma, Julie A Adams, and Chen Zhou. An agent-based architecture for an adaptive human-robot interface. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 8–pp. IEEE, 2003.

[102] Richard Kennard and RJ Steele. Application of software mining to automatic user interface generation. In *International Conference on Software Methods and Tools*. IOS Press, 2008.

[103] Marius Kloetzer and Cristian Mahulea. A petri net based approach for multi-robot path planning. *Discrete Event Dynamic Systems*, 24(4):417–445, 2014.

[104] Andrew J Ko, Brad A Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 199–206. IEEE, 2004.

[105] Kenneth R Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew Hockenberry. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent tutoring systems*, volume 3220, pages 7–10. Springer, 2004.

[106] Stefan Kohlbrecher, Alberto Romay, Alexander Stumpf, Anant Gupta, Oskar Von Stryk, Felipe Bacim, Doug A Bowman, Alex Goins, Ravi Balasubramanian, and David C Conner. Human-robot teaming for rescue missions: Team vigir's approach to the 2013 darpa robotics challenge trials. *Journal of Field Robotics*, 32(3):352–377, 2015.

[107] Stefan Kopp, Brigitte Krenn, Stacy Marsella, Andrew N Marshall, Catherine Pelachaud, Hannes Pirker, Kristinn R Thórisson, and Hannes Vilhjálmsson. Towards a common framework for multimodal generation: The behavior markup language. In *International Workshop on Intelligent Virtual Agents*, pages 205–217. Springer, 2006.

[108] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[109] David Kortenkamp, Reid Simmons, and Davide Brugali. Robotic systems architectures and programming. In *Springer Handbook of Robotics*, pages 283–306. Springer International Publishing, 2016.

[110] Yehia Thabet Kotb, Steven S Beauchemin, and John L Barron. Petri net-based cooperation in multi-agent systems. In *CRV*, pages 123–130, 2007.

[111] James Kramer and Matthias Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22(2):101–132, 2007.

[112] Geert-Jan M Kruijff, M Janíček, Shanker Keshavdas, Benoit Larochelle, Hendrik Zender, Nanja JJM Smets, Tina Mioch, Mark A Neerincx, Jurriaan Van Diggelen, Francis Colas, et al. Experience in system design for human-robot teaming in urban search and rescue. In *Field and Service Robotics*, pages 111–125. Springer, 2014.

[113] Geert-Jan M Kruijff, Fiora Pirri, Mario Gianni, Panagiotis Papadakis, Matia Pizzoli, Arnab Sinha, Viatcheslav Tretyakov, Thorsten Linder, Emanuele Pianese, Salvatore Corrao, et al. Rescue robots at earthquake-hit mirandola, italy: A field report. In *Safety, security, and rescue robotics (SSRR), 2012 IEEE international symposium on*, pages 1–8. IEEE, 2012.

[114] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. Tradr project: Long-term human-robot teaming for robot assisted disaster response. *KI-Künstliche Intelligenz*, 29(2):193–201, 2015.

[115] Gerard Lacey and Shane MacNamara. Context-aware shared control of a robot mobility aid for the elderly blind. *The International Journal of Robotics Research*, 19(11):1054–1065, 2000.

[116] Michail G Lagoudakis, Marc Berhault, Sven Koenig, Pinar Keskinocak, and Anton J Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 698–705. IEEE, 2004.

[117] Leslie Lamport. *I1 (\ LaTeX)—A Document*, volume 410. pub-AW, 1985.

[118] Pat Langley. Machine learning for adaptive user interfaces. In *KI-97: Advances in artificial intelligence*, pages 53–62. Springer, 1997.

[119] Jin-Shyan Lee, Meng-Chu Zhou, and Pau-Lo Hsu. An application of petri nets to supervisory control for human-computer interactive systems. *IEEE Transactions on Industrial Electronics*, 52(5):1220–1226, 2005.

[120] Pedro Miguel Amado Rodrigues Leonardo. *Child programming: an adequate domain specific language for programming specific robots.* PhD thesis, Faculdade de Ciências e Tecnologia, 2013.

[121] Hector J Levesque, Philip R Cohen, and José HT Nunes. On acting together. In *AAAI*, volume 90, pages 94–99, 1990.

[122] James R Lewis. Sample sizes for usability studies: Additional considerations. *Human factors*, 36(2):368–378, 1994.

[123] Diego Lirman, Nuno Ricardo Gracias, Brooke Erin Gintert, Arthur Charles Rogde Gleason, Ruth Pamela Reid, Shahriar Negahdaripour, and Philip Kramer. Development and application of a video-mosaic survey technology to document the status of coral reef communities. *Environmental monitoring and assessment*, 125(1):59–73, 2007.

[124] Lantao Liu and Dylan A Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33(3):291–307, 2012.

[125] Bingxian Ma. Modeling multi-agent systems with hierarchical colored petri nets. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 167–171. Springer, 2005.

[126] Douglas C MacKenzie, Jonathan M Cameron, and Ronald C Arkin. Specification and execution of multiagent missions. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 3, pages 51–58. IEEE, 1995.

[127] Louis Major, Theocharis Kyriacou, and O Pearl Brereton. Systematic literature review: teaching novices programming using robots. *IET software*, 6(6):502–513, 2012.

[128] Limor Marciano. Cpnp: Colored petri net represention of single-robot and multi-robot plans. Master's thesis, Citeseer, 2013.

[129] Alessandro Marino, Lynne Parker, Gianluca Antonelli, and Fabrizio Caccavale. Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 831–836. IEEE, 2009.

[130] Maja J Matarić, Gaurav S Sukhatme, and Esben H Østergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2-3):255–263, 2003.

[131] Ivan Maza, Fernando Caballero, Jesus Capitan, José Ramiro Martinez-de Dios, and Anibal Ollero. A distributed architecture for a robotic platform with aerial sensor transportation and self-deployment capabilities. *Journal of Field Robotics*, 28(3):303–328, 2011.

[132] Juan Pablo Mendoza, Manuela Veloso, and Reid Simmons. Detecting and correcting model anomalies in subspaces of robot planning domains. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1587–1595. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[133] Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Distributed multi-robot task assignment and formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 128–133. IEEE, 2008.

[134] Mark Micire, Munjal Desai, Amanda Courtemanche, Katherine M Tsui, and Holly A Yanco. Analysis of natural gestures for controlling robot teams on multi-touch tabletop surfaces. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 41–48. ACM, 2009.

[135] C Miller, R Goldman, H Funk, Peggy Wu, and B Pate. A playbook approach to variable autonomy control: Application for control of multiple, heterogeneous unmanned air vehicles. In *Proceedings of FORUM 60, the Annual Meeting of the American Helicopter Society*, pages 7–10, 2004.

[136] Christopher Miller, Harry Funk, Peggy Wu, Robert Goldman, John Meisner, and Marc Chapman. The playbook^{TM} approach to adaptive automation. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 49, pages 15–19. SAGE Publications, 2005.

[137] Christopher A Miller and Raja Parasuraman. Designing for flexible interaction between humans and automation: Delegation interfaces for supervisory control. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 49(1):57–75, 2007.

[138] Daniel Moody. The physics of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009.

[139] Lilia Moshkina, Yoichiro Endo, and Ronald C Arkin. Usability evaluation of an automated mission repair mechanism for mobile robot mission specification. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 57–63. ACM, 2006.

[140] Robin R Murphy. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(2):138–153, 2004.

[141] Vignesh Narayanan, Yu Zhang, Nathaniel Mendoza, and Subbarao Kambhampati. Automated planning for peer-to-peer teaming and its evaluation in remote human-robot interaction. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*, pages 161–162. ACM, 2015.

[142] Hoa G Nguyen, Narek Pezeshkian, Anoop Gupta, and Nathan Farrington. Maintaining communication link for a robot operating in a hazardous environment. Technical report, SPACE AND NAVAL WARFARE SYSTEMS COMMAND SAN DIEGO CA, 2004.

[143] Jakob Nielsen, Ida Frehr, and HANS OLAV NYMAND. The learnability of hypercard as an object-oriented programming system. *Behaviour & Information Technology*, 10(2):111–120, 1991.

[144] Mie Nørgaard and Kasper Hornbæk. What do usability evaluators do in practice?: an explorative study of think-aloud testing. In *Proceedings of the 6th conference on Designing Interactive systems*, pages 209–218. ACM, 2006.

[145] Steven Okamoto, Nathan Brooks, Sean Owens, Katia Sycara, and Paul Scerri. Allocating spatially distributed tasks in large, dynamic robot teams. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1245–1246. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[146] Dan R Olsen and Michael A Goodrich. Metrics for evaluating human-robot interactions. In *Proceedings of PERMIS*, volume 2003, page 4, 2003.

[147] Raja Parasuraman, Keryl A Cosenzo, and Ewart De Visser. Adaptive automation for human supervision of multiple uninhabited vehicles: Effects on change detection, situation awareness, and mental workload. *Military Psychology*, 21(2):270, 2009.

[148] Raja Parasuraman, Scott Galster, Peter Squire, Hiroshi Furukawa, and Christopher Miller. A flexible delegation-type interface enhances system performance in human supervision of multiple robots: Empirical studies with roboflag. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(4):481–493, 2005.

[149] Raja Parasuraman, Thomas B Sheridan, and Christopher D Wickens. A model for types and levels of human interaction with automation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(3):286–297, 2000.

[150] James Lyle Peterson. Petri net theory and the modeling of systems. *PRENTICE-HALL, INC., ENGLEWOOD CLIFFS, NJ 07632, 1981, 290*, 1981.

[151] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 46–51. IEEE, 2009.

[152] Amanda Prorok, M Ani Hsieh, and Vijay Kumar. Fast redistribution of a swarm of heterogeneous robots. In *proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) on 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 249–255. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[153] David V Pynadath, Paul Scerri, and Milind Tambe. Towards adjustable autonomy for the real world. *arXiv preprint arXiv:1106.4573*, 2011.

[154] David V Pynadath, Milind Tambe, Nicolas Chauvat, and Lawrence Cavedon. Toward team-oriented programming. In *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, pages 233–247. Springer, 2000.

[155] Morgan Quigley, Blake Barber, Steve Griffiths, and Michael A Goodrich. Towards real-world searching with fixed-wing mini-uavs. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3028–3033. IEEE, 2005.

[156] Dave Raggett, Arnaud Le Hors, Ian Jacobs, et al. Html 4.01 specification. *W3C recommendation*, 24, 1999.

[157] Sarvapali D Ramchurn, Trung Dong Huynh, Yuki Ikuno, Jack Flann, Feng Wu, Luc Moreau, Nicholas R Jennings, Joel E Fischer, Wenchao Jiang, Tom Rodden, et al. Hac-er: a disaster response system based on human-agent collectives. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 533–541. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[158] Jean-François Raskin and Laurent Van Begin. Petri nets with non-blocking arcs are difficult to analyze. *Electronic Notes in Theoretical Computer Science*, 98:35–55, 2004.

[159] 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.

[160] Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008.

[161] Pierre-Alain Reynier and Frédéric Servais. Minimal coverability set for petri nets: Karp and miller algorithm with pruning. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 69–88. Springer, 2011.

[162] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. Coordination and control of multiple uavs. In *AIAA guidance, navigation, and control conference, Monterey, CA*, 2002.

[163] Max Risler and Oskar von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl. In *AAMAS08-workshop on formal models and methods for multi-robot systems, Estoril, Portugal*. Citeseer, 2008.

[164] Juan Jesús Roldán, Jaime del Cerro, and Antonio Barrientos. A proposal of methodology for multi-uav mission modeling. In *Control and Automation (MED), 2015 23th Mediterranean Conference on*, pages 1–7. IEEE, 2015.

[165] Ariel Rosenfeld, Noa Agmon, Oleg Maksimov, Amos Azaria, and Sarit Kraus. Intelligent agent supporting human-multi-robot team collaboration. In *IJCAI*, pages 1902–1908, 2015.

[166] Heath A Ruff, Sundaram Narayanan, and Mark H Draper. Human interaction with levels of automation and decision-aid fidelity in the supervisory control of multiple simulated unmanned air vehicles. *Presence: Teleoperators and virtual environments*, 11(4):335–351, 2002.

[167] Malcolm Ross Kinsella Ryan. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research*, 31:497–542, 2008.

[168] Paul M Salmon, Neville A Stanton, Guy H Walker, Daniel Jenkins, Darshna Ladva, Laura Rafferty, and Mark Young. Measuring situation awareness in complex systems: Comparison of measures study. *International Journal of Industrial Ergonomics*, 39(3):490–500, 2009.

[169] 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 *International Conference on Autonomous Agents and Multiagent Systems*, pages 252–269. Springer, 2011.

[170] Paul Scerri, David V Pynadath, Nathan Schurr, Alessandro Farinelli, Sudeep Gandhe, and Milind Tambe. Team oriented programming and proxy agents: The next generation. In *International Workshop on Programming Multi-Agent Systems*, pages 131–148. Springer, 2003.

[171] Paul Scerri, Prasanna Velagapudi, Balajee Kannan, Abhinav Valada, Christopher Tomaszewski, John Dolan, Adrian Scerri, Kumar Shaurya Shankar, Luis Bill, and George Kantor. Real-world testing of a multi-robot team. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1213–1214. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[172] Jean Scholtz, Jeff Young, Jill L Drury, and Holly A Yanco. Evaluation of human-robot interaction awareness in search and rescue. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2327–2332. IEEE, 2004.

[173] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, DTIC Document, 1978.

[174] Reid Simmons. Contextual awareness for robust robot autonomy. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA, 2013.

[175] Glenn Smith, Robert Smith, and Aster Wardhani. Software reuse across robotic platforms: Limiting the effects of diversity. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 252–261. IEEE, 2005.

[176] Rick Spencer. The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 353–359. ACM, 2000.

[177] Aaron Steinfeld. Interface lessons for fully and semi-autonomous mobile robots. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2752–2757. IEEE, 2004.

[178] David B Stewart and Pradeep K Khosla. Rapid development of robotic applications using component-based real-time software. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 465–470. IEEE, 1995.

[179] Kartik Talamadupula, Gordon Briggs, Tathagata Chakraborti, Matthias Scheutz, and Subbarao Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2957–2962. IEEE, 2014.

[180] Milind Tambe. Towards flexible teamwork. *arXiv preprint cs/9709101*, 1997.

[181] Kimmo Tarkkanen, Pekka Reijonen, Franck Tétard, and Ville Harkke. Back to user-centered usability testing. In *Human Factors in Computing and Informatics*, pages 91–106. Springer, 2013.

[182] Paul Taylor and Amy Isard. Ssml: A speech synthesis markup language. *Speech communication*, 21(1-2):123–133, 1997.

[183] Christopher Tomaszewski, Abhinav Valada, and Paul Scerri. Planning efficient paths through dynamic flow fields in real world domains. In *Oceans-San Diego, 2013*, pages 1–5. IEEE, 2013.

[184] Craig Tovey, Michail G Lagoudakis, Sonal Jain, and Sven Koenig. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 3–14. Springer, 2005.

[185] Patrick Ulam, Yoichiro Endo, Alan Wagner, and Ronald Arkin. Integrated mission specification and task allocation for robot teams-design and implementation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4428–4435. IEEE, 2007.

[186] Abhinav Valada, Christopher Tomaszewski, Balajee Kannan, Prasanna Velagapudi, George Kantor, and Paul Scerri. An intelligent approach to hysteresis compensation while sampling using a fleet of autonomous watercraft. In *Intelligent Robotics and Applications*, pages 472–485. Springer, 2012.

[187] 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.

[188] MW van de Nes. Coverability and extended petri nets. B.s. thesis, Universiteit Leiden, 2013.

[189] Lovekesh Vig and Julie A Adams. Market-based multi-robot coalition formation. *Distributed Autonomous Robotic Systems 7*, pages 227–236, 2006.

[190] Petcharat Viriyakattiyaporn and Gail C Murphy. Challenges in the user interface design of an ide tool recommender. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 104–107. IEEE Computer Society, 2009.

[191] Alan Wagner, Yoichiro Endo, Patrick Ulam, and Ronald Arkin. Multi-robot user interface modeling. *Distributed Autonomous Robotic Systems 7*, pages 237–248, 2006.

[192] Fei-Yue Wang, Yanqing Gao, and MengChu Zhou. A modified reachability tree approach to analysis of unbounded petri nets. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):303–308, 2004.

[193] Huadong Wang, Andreas Kolling, Nathan Brooks, Sean Owens, Shafiq Abedin, Paul Scerri, Pei-ju Lee, Shih-Yi Chien, Michael Lewis, and Katia Sycara. Scalable target detection for large robot teams. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 363–370. ACM, 2011.

[194] Huadong Wang, Michael Lewis, Prasanna Velagapudi, Paul Scerri, and Katia Sycara. How search and its subtasks scale in n robots. In *Human-Robot Interaction (HRI), 2009 4th ACM/IEEE International Conference on*, pages 141–147. IEEE, 2009.

[195] David Wettergreen, Nathalie Cabrol, Vijayakumar Baskaran, Francisco Calderón, Stuart Heys, Dominic Jonak, R Allan Luders, David Pane, Trey Smith, James Teza, et al. Second experiments in the robotic investigation of life in the atacama desert of chile. In *Proc. 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[196] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. The cognitive walkthrough method: A practitioner's guide. In *Usability inspection methods*, pages 105–140. John Wiley & Sons, Inc., 1994.

[197] 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.

[198] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12):399, 2013.

[199] Tao Zhang and Julie A Adams. Evaluation of a geospatial annotation tool for unmanned vehicle specialist interface. *International Journal of Human-Computer Interaction*, 28(6):361–372, 2012.

[200] Anmin Zhu and Simon X Yang. A neural network approach to dynamic task assignment of multirobots. *IEEE transactions on neural networks*, 17(5):1278–1287, 2006.

[201] Vittorio A Ziparo, Luca Iocchi, Pedro U Lima, Daniele Nardi, and Pier Francesco Palamara. Petri net plans. *Autonomous Agents and Multi-Agent Systems*, 23(3):344–383, 2011.

Appendices

.1 Source code

Java source code for the SPN language and CRW simulator, sample plans, and GUI are available on GitHub.
`https://github.com/nbbrooks/sami-core`
`https://github.com/nbbrooks/sami-dreaam`
`https://github.com/nbbrooks/sami-crw`

.2 CRW library

.2.1 Output event list

Name	**Description**	**Fields**	**Uses Proxy?**	**Uses Task?**
Proxy Directives				
Proxy Set Description	Set a description for the proxy	String (description)	Y	N
Proxy Execute Path	Tell the proxy(ies) to follow a path	Map <Proxy, Path >	Y	N
Proxy Execute Path and Block	Tell the proxy(ies) to follow a path and wait	Map <Proxy, Path >	Y	N
Proxy Goto Point	Tell the proxy(ies) to go directly (straight-line) to a location	Map <Proxy, Location >	Y	N
Proxy Goto Point and Block	Tell the proxy(ies) to go directly to a location and wait	Map <Proxy, Location >	Y	N
Proxy Explore Area	Tell the proxy(ies) to collaboratively explore an area	Area2D, double (Mapping density)	Y	N
Proxy Resend Waypoints	Resend a proxy's current waypoints	-	Y	N
Connect Existing Proxy	Connect to a boat server via IP address		Y	N
Connect Existing Proxy ID	Connect to a boat server via ID		Y	N
Create Simulated Proxy	Create and connect to a simulated boat		Y	N
Set Gains	Set the proxy(ies)'s PID gains for steering and throttle	List¡double¿ (Gains)	Y	N
Block Movement	Tell the proxy(ies) to wait for further instruction from this mission	-	Y	N

Name	**Description**	**Fields**	**Uses Proxy?**	**Uses Task?**
Proxy Goto LatLon	Tell the proxy(ies) to go directly (straight-line) to a location	Location	Y	N
Service Requests				
Allocation Request	Get possible allocations of a set of tasks to a set of proxies	-	Y	Y
Path Request	Get path from proxy's location to a location	Map <Proxy, Location >	Y	N
Assemble Location Request	Give each proxy a unique location near a location	Location (Central location), double (Spacing density in m)	Y	N
Proxy Compare Distance Request	Compare proxy's distance to a point	Location (Comparison location), double (Distance threshold in m)	Y	N
Start System Timer	System wide timer	int (Duration in ms)	N	N
Start Proxy Timer	Proxy specific timer	int (Duration in ms)	Y	N
Get All Proxy Tokens	Get all proxy tokens	-	N	N
Battery Level Subscription	Battery level subscription	int (Update rate in s), double (Low battery % threshold), double (Critical battery % threshold)	Y	N
UI Directives				
Display Message	GUI message	Text Message	N	N
Operator Allocation Options	Select from allocations	List <Allocation >	N	N
Operator Path Options	Select from paths	List <Map <Proxy, Path >>	N	N

Name	**Description**	**Fields**	**Uses Proxy?**	**Uses Task?**
Operator Select Boat	Select from proxy tokens	-	Y	N
Operator Select Boat List	Select from proxy tokens	-	Y	N
Operator Select Boat ID	Select from in scope boat IDs	List <BoatID >	N	N
Operator Create Area	Create area	-	N	N
Operator Approve	Grant approval	-	N	N
Variables				
Define Variables	List of variables to define	List¡Variable¿	N	N
Redefine Variables	Variable name to redefine	VariableName	N	N
Select Variable	Returns list of in-scope variables of specified class	VariableClass	N	N
Return Value	Set's sub-mission "return" String value	String	N	N
Abort				
Send Abort Mission	Tell proxy to send signal to abort current mission	-	Y	N
Send Proxy Abort All Missions	Tell proxy to send signal to abort all missions	-	Y	N
Send Proxy Abort Future Missions	Tell proxy to send signal to abort future missions	-	Y	N
Proxy Abort Mission	Proxy aborts actions related to mission	-	Y	N
Tasks				
Generate Tasks	Create a Send TaskStarted for each proxy's current task	List¡Location¿, TaskClass	N	Y
Refresh Tasks	Send TaskStarted for each proxy's current task	-	N	Y
Task Complete	Signal that the task is complete	-	N	Y

Table 1: Summary of CRW output events

.2.2 Input event list

Name	**Description**	**Fields**	**Uses RT?**
Proxy			
Proxy Path Completed	Proxy has completed its path	-	Y
Proxy Created	Connection to boat has been established and a proxy token has been created	-	Y
Proxy Pose Updated	Proxy has received a pose update from its boat	-	Y
Service Responses			
Allocation Response	Allocation algorithm has returned computed allocation(s) for a set of tasks	List <Allocation >	Y
Path Response	Path algorithm has returned computed path(s) for a set of proxies	List <Map <Proxy, Path >>	Y
Battery Nominal	Proxy's battery level is at/above configured "nominal" level	double (Battery %)	Y
Battery Low	Proxy's battery level is at/above configured "low" level and below "nominal" level	double (Battery %)	Y
Battery Critical	Proxy's battery level is below configured "low" level	double (Battery %)	Y
Assemble Location Response	Spaced out proxy locations have been computed	Map <Proxy, Location >	Y
Timer Expired	System timer has expired	-	N
Proxy Timer Expired	Proxy's timer has expired	-	Y
Tokens Returned	A request for a set of tokens has been completed	-	Y
Quantity Greater	The first of two quantities compared was greater	-	Y
Quantity Less	The second of two quantities compared was greater	-	Y
Quantity Equal	The two quantities compared were equal	-	Y
Selection			
Operator Accepted Allocation	One of the presented allocation options was accepted	Allocation	Y
Operator Rejected Allocation	The presented allocation options was rejected	-	Y
Operator Accepts Path	One of the presented path options was accepted	Map <Proxy, Path >	Y
Operator Rejects Path	The presented path options was rejected	-	Y
Operator Selects Boat	One of the presented proxies was selected	-	Y

Name	**Description**	**Fields**	**Uses RT?**
Operator Selects Boat List	One/more of the presented proxies was selected	-	Y
Operator Selects Boat ID	One of the presented proxies IDs was selected	BoatId	Y
Operator Created Area	Operator defined a 2D area	Area2D	N
Yes Option	Operator responded "yes"	-	N
No Option	Operator responded "no"	-	N
Variables			
Defined Variables Received	Operator defined the requested variables	Map <String, Object >	N
Redefined Variables Received	Operator redefined the requested variables	Map <String, Object >	N
Variable Selected	Operator selected one of the presented variables	VariableReference	N
Check Return	Check sub-mission return value against a string	MissionPlanSpecification (sub-mission), String (Comparison value)	N
Abort			
Proxy Abort Mission Received	The proxy signalled to abort this mission	-	Y
Operator Interrupt Received	The operator triggered an interrupt	String (Interrupt name)	N
Tasks			
Task Delayed	This task has moved backwards in the assigned proxy's queue	-	Y
Task Reassigned	This task has been reassigned to another proxy	-	Y
Task Released	This proxy is no longer responsible for this task	-	Y
Task Started	This task has been started by its proxy	-	Y
Task Unassigned	This task is not assigned to any proxy	-	Y

Table 2: Summary of CRW input events

.2.3 Markup list

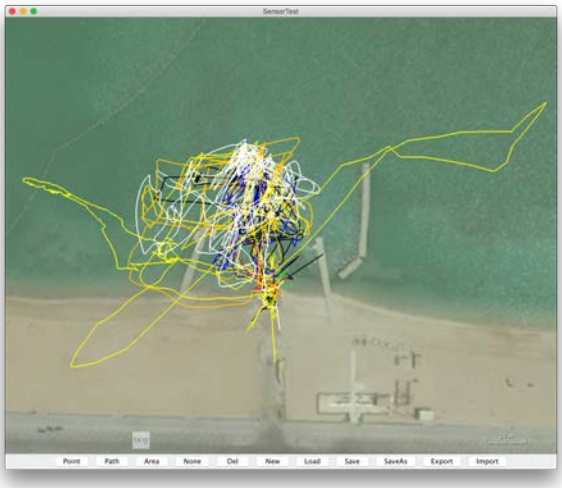
Name	Description	Options
Attention	Attract the operator's decision	**Type** {Highlight / Blink}, **End** {On Click}, **Target** {Component / All proxies / Relevant proxies}
Default Selection	Set a default option for a decision	**Selection** {All / None}
Description	Add additional text information to a decision	**Source** {Specify text}, **Show plan name** {Yes / No}, **Show vertex name** {Yes / No}, **Show event name** {Yes / No}
Filter	Filter to include/exclude certain options in a decision	**Source** {Specify text}, **Action** {Include / Exclude}
Keyword	Attach a keyword for grouping similar interactions	**Source** {Specify text}
Mixed Initiative Trigger	Customize MI strategy for this decision	**Trigger** {Never / Immediately / Timer}
Selection Options	Specify the number of solutions that should be computed and presentation style	**Number** {Specify number}, **Format** {Sequential / Stacked / Tabs}
Optimize	Provide hints for what algorithms should optimize for	**Criteria** {Computation time / Execution time / Execution safety / Execution efficiency}
Phase Branch	Indicate this vertex branches the previous phase item into the next phase item	**Action name source** {Specify text}, **Actor** {Operator / SAMI / Proxy / Task}
Phase Item	Indicate this vertex is a member of the indicated phase	**Phase name source** {Specify text}, **Actor** {Operator / SAMI / Proxy / Task}
Priority	Specify a relative priority for this interaction	**Rank** {Low / Medium / High / Critical}
Proxy Status	Specify the proxy's status in this mission	**Status** {Nominal, Warning, Severe}
Relevant Area	Specify an area relevant to this interaction	**Source** {Specify Area, Specify point, All proxies, Relevant proxies}, **Modification** {Expand / Reduce}, **Map type** {Satellite / Political}
Relevant Information	Specify a data type relevant to this interaction	**Source** {Select sensor}, **Visuzalition** {Heatmap, Contour, Threshold}
Relevant Proxy	Specify proxy(ies) relevant to this interaction	**Proxy** {Relevant proxies / All proxies}, **Show path** {Yes / No}, **Show diagnostic** {Yes / No}, **Show data** {Yes / No}

Name	Description	Options
Sort	Sort options by some criteria	**Method** {Alphabetical / Chronological}, **Order** {Increasing / Decreasing}

Table 3: Summary of CRW markups

.3 Field deployment data

(a) Boat GPS logs for December 13, 2015



(b) Boat GPS logs for December 15, 2015

(c) Boat GPS logs for December 17, 2015

(d) Boat GPS logs for December 19, 2015

.4 User study material

.4.1 DREAMM Lesson Modes

Action	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**	**L10**	**L11**	**L12**	**L13**
Edit label												
Edit events												
Delete												
Edit sub-missions												
Set/unset start												
Set/unset end												
Edit in/out tokens												
Delete in/out edge												
Blank place												
Places												
Blank transition												
Transitions												
Roles												
Sub-missions												
Save												
Save As												
Exit												
Clear all												
Edit Markup												
Read variable name												
Value												
Write variable name												
Editable												
Clear all												
Edit Markup												
Write variable name												
No edge requirement icon												
Error counter												
Warning counter												
Visible												
New mission												
Rename												
Delete												

Action	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**	**L10**	**L11**	**L12**	**L13**
Visible												
AnyProxy												
AnyTask												
Implied Generic												
Generic												
RelevantToken												
TaskClass												
None												
LessThan												
GreaterThanEqualTo												
AnyProxy												
AnyTask												
Implied Generic												
Generic												
RelevantToken												
TaskClass												
All												
Number												
Take												
Consume												
Add												
Save												
Undo												
Redo												

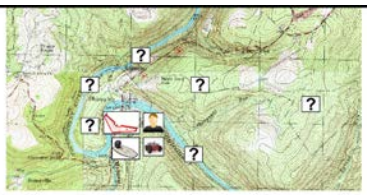
Table 4: DREAMM lesson modes

.4.2 Tutorial Slides

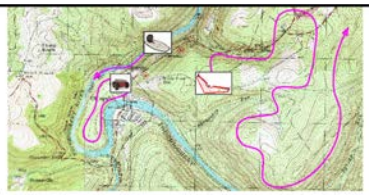
Introduction

When teams of humans and robots coordinate their efforts, they can use their individual strengths to achieve a common goal they could not achieve individually.

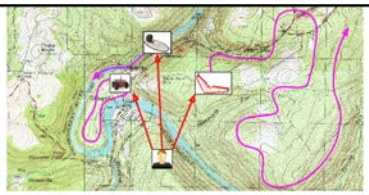
In these lessons, you will learn a language used to describe team plans: the tasks for each team member to achieve a shared goal.



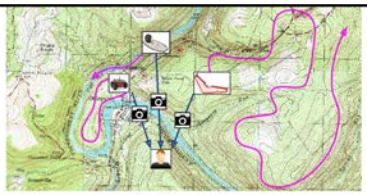
Consider a scenario where a human-robot team needs to investigate areas for signs of a lost hiker.



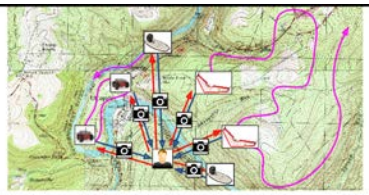
The robots have various sensing and movement capabilities which, when coordinated via a team plan, can rapidly cover an area.



A human can help search, but also has a greater understanding of the nuances of the overall goal and is better used in the team plan as an "operator" who provides high level instructions to the robots.



While the robots have some capability to understand their sensors, the human again has a greater understanding of an image or recording. So in this scenario we would want information to be received by the human, and allow them to adjust the robot's responsibilities.



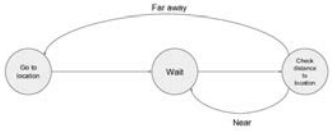
As more robots are added to the team, the operator can easily be overwhelmed by the amount of information and robot instruction they are responsible for. To address this, the language allows for specific parts of a team plan to be "marked up" with strategies to reduce the operator's workload.

Intro to SAMI Petri Nets (SPNs)

The language you will learn is named "SAMI Petri Nets", or SPN, which is based on the based on the "Petri Net" mathematical modeling language. In this lesson you will learn about the building blocks of the Petri Net language.

We will learn the language by incrementally building up an example team plan, adding features of the language to the team plan as we learn them.

Let's start with a team plan that only considers one robot and no humans. Consider the goal of having the robot "station keep" around a location, periodically checking if it has drifted away from its location and moving back if necessary. Drift could be caused by water current, terrain slope, wind, or many other factors. The logic for such behavior would look something like this:



Now we will begin to represent the diagram using a SAMI Petri Net (SPN).

3 key building blocks of SPN are:

- Places: describe the status of the system

Represented by circles

Now we will begin to represent the diagram using a SAMI Petri Net (SPN).

3 key building blocks of SPN are:

- Places: describe the status of the system
- Transitions: describe a change in the system

Represented by vertical lines

Now we will begin to represent the diagram using a SAMI Petri Net (SPN).

3 key building blocks of SPN are:

- Places: describe the status of the system
- Transitions: describe a change in the system
- Edges: describe how a change affects the status of the system
  - In Edges connect a place to a transition

Represented by arrows

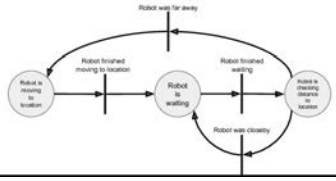
Now we will begin to represent the diagram using a SAMI Petri Net (SPN).

3 key building blocks of SPN are:

- Places: describe the status of the system
- Transitions: describe a change in the system
- Edges: describe how a change affects the status of the system
  - In Edges connect a place to a transition
  - Out Edges connect a transition to a place

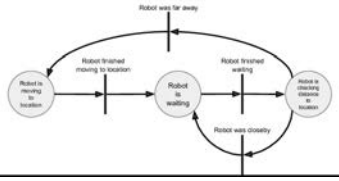
Represented by arrows

The final set of places, transitions, and edges would look like this

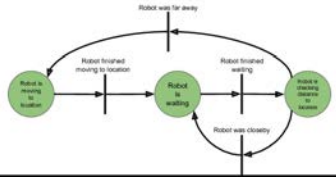


Now you will answer some questions about the SPN language to evaluate how well the lessons have worked. When a quiz shows up, fill in your answer on the corresponding print out. Then you can go to the next slide, which contains the answer. After the lesson is complete, we will briefly discuss the questions.

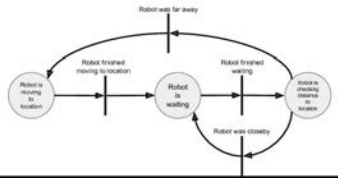
Quiz 2-1: Identify the places



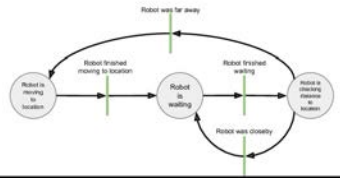
Quiz 2-1 Solution



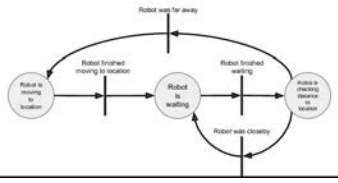
Quiz 2-2: Identify the transitions



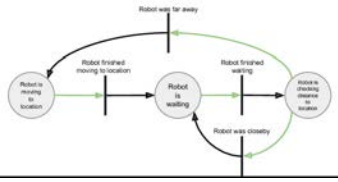
Quiz 2-2 Solution



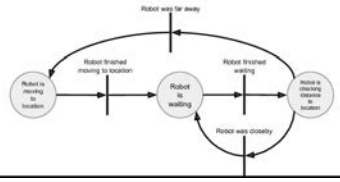
Quiz 2-3: Identify the in edges



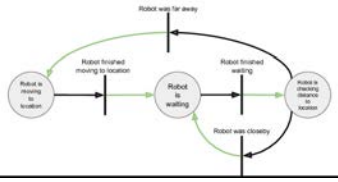
Quiz 2-3 Solution



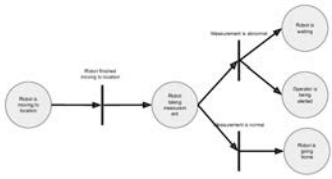
Quiz 2-4: Identify the out edges



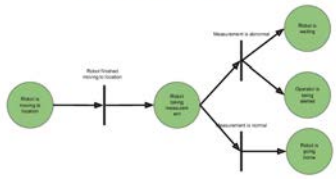
Quiz 2-4 Solution



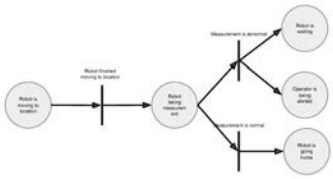
Quiz 2-5: Identify the places



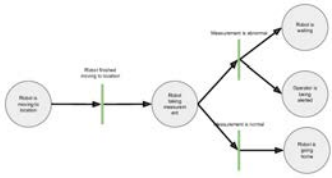
Q2-5 Solution



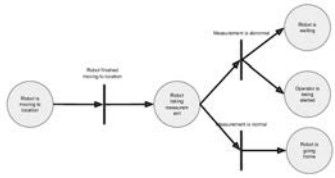
Quiz 2-6: Identify the transitions



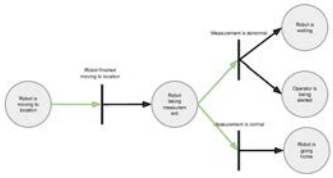
Quiz 2-6 Solution



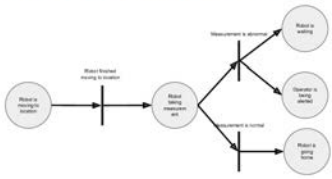
Quiz 2-7: Identify the in edges



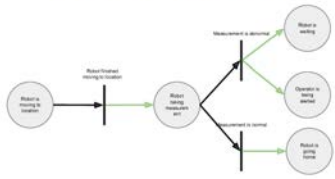
Quiz 2-7 Solution



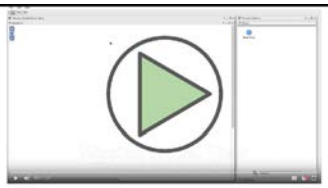
Quiz 2-8: Identify the out edges



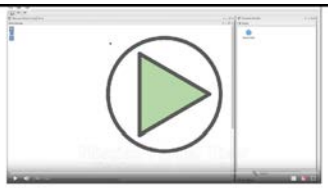
Quiz 2-8 Solution



Watch "UI Overview": This video gives you an overview of DREAMM: the program you will use to develop SPN team plans.



Watch "Places, Transitions, and Edges": This video will show you how to add and manipulate places, transitions, and edges to a SPN.

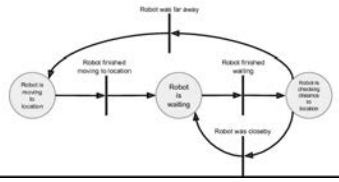


Watch "Multiple SPNs": This video will show you how to switch between different SPNs.

Now you will practice creating some SPNs in DREAMM from figures like you've seen in this lesson. When constructing these, use the text on the place/transition in the figure as the label for the place/transition in DREAMM.

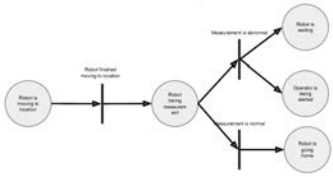
Job 2-1: Create this SPN in DREAMM

Name the SPN job2-1



Job 2-2: Create this SPN in DREAMM

Name the SPN job2-2



Events

So far we have just written descriptions on places and transitions, such as "Robot is moving to location" and "Robot was far away." Events are what make things actually move and do work.

There are 2 types of events

- Output events: Are put on places and contain requests/commands which are sent to robots or the operator.
- Input events: Are put on transitions and contain information which was sent by a robot or the operator.

Output events: Are put on places and contain requests/commands which are sent to team members, such as robots or the operator.

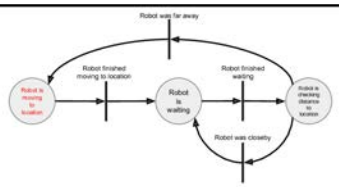
Below are some example output events. We put a right-facing arrow at the beginning of its name to indicate it is an output event.

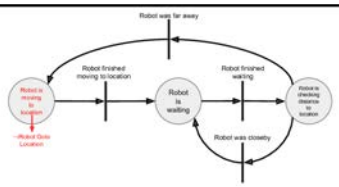
→Robot Goto Location: commands the robot to go to a specified location

→Robot Start Timer: commands the robot to wait for a specified amount of time

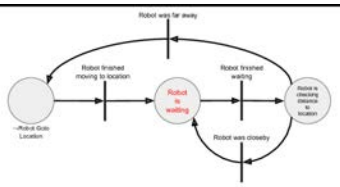
→Robot Compare Distance: commands the robot to compare its distance from a location to a specified distance

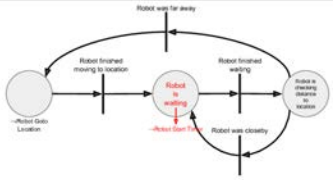
Now we will go through our station keep example and replace the description labels on places with the output events that will actually make those things happen.



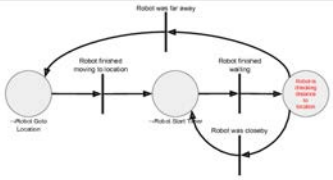


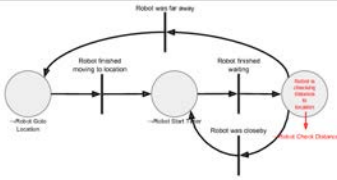
→RobotGotoLocation: commands the robot to go to a specified location





→RobotStartTimer: commands the robot to wait for a specified amount of time

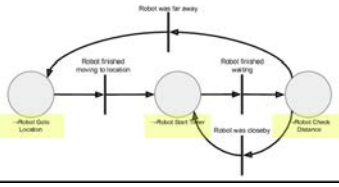




→RobotCompareDistance: commands the robot to compare its distance from a location to a specified distance

Watch "Output Events": This video will show you how to add Output Events to Places.

Job 3-1: Add each of the output events to the SPN



Input events: Are put on transitions and contain information sent by a team-member, such as a robot, an AI (artificial intelligence), or the operator.

The input events are generated when the robot, AI, or operator sends information.

Below are some examples of input events. We put a left-facing arrow at the beginning of its name to indicate it is an input event.

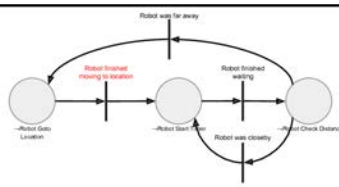
←Robot at Location: the robot has arrived at its assigned location

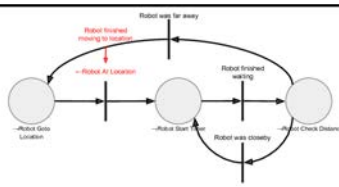
←Robot Timer Expired: the robot's timer has expired

←Quantity Greater: the first of two numbers compared by an AI was larger

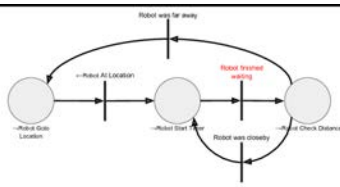
←Quantity Lesser: the first of two numbers compared by an AI was lesser

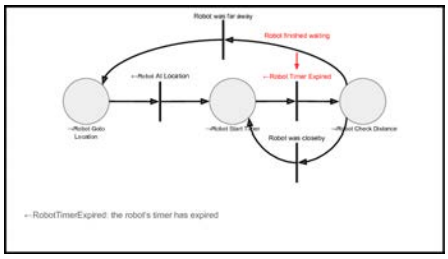
←Quantity Equal: the two numbers compared by an AI were equal

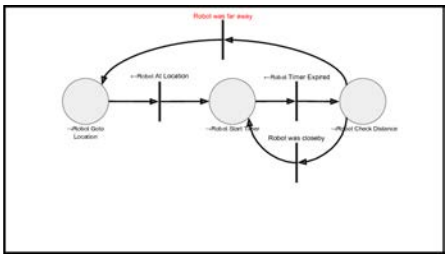


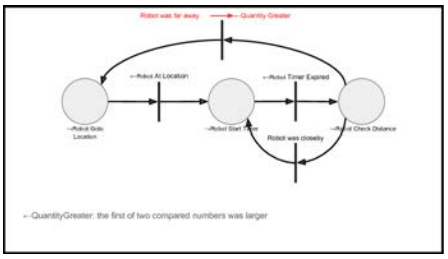


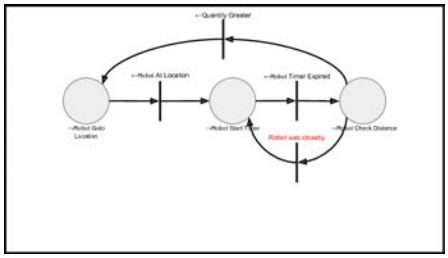
←RobotAtLocation: the robot has arrived at its assigned location

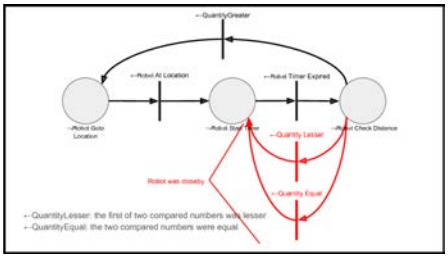


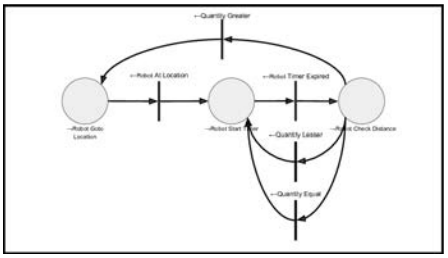


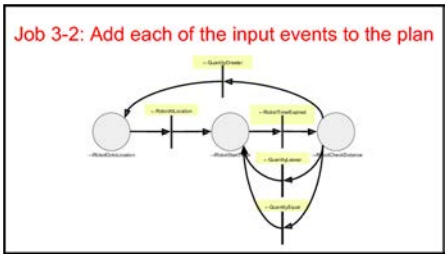


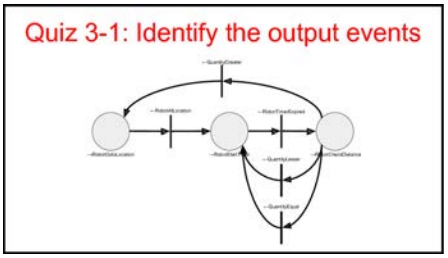


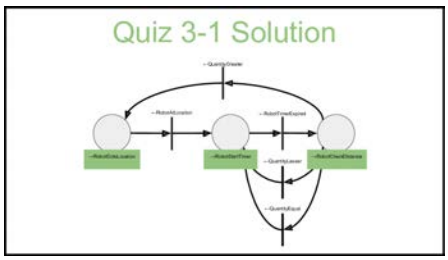


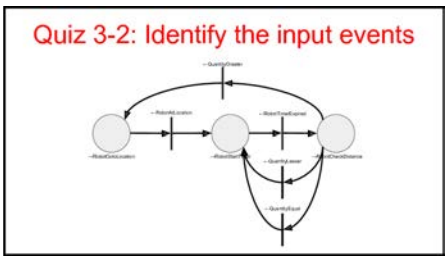


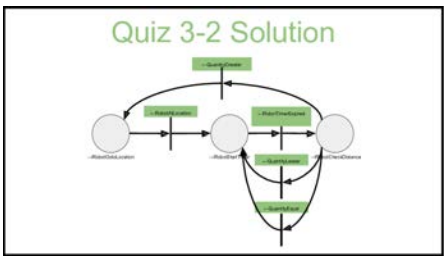


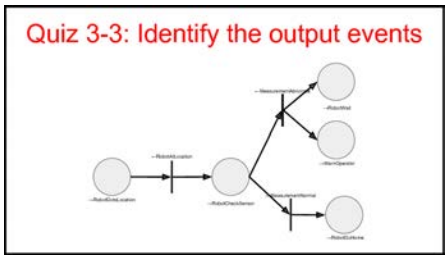


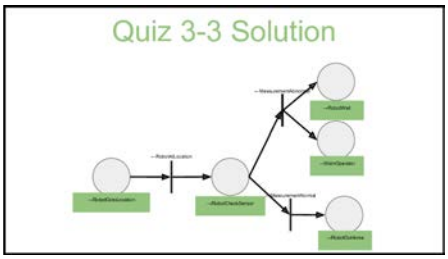


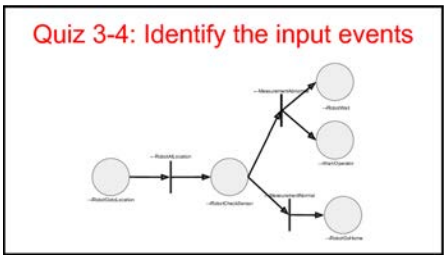


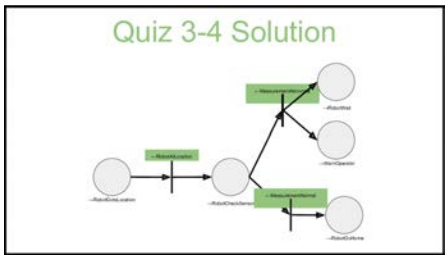












With the addition of output events we can send commands to the robot.

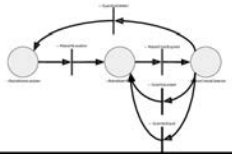
With the addition of input events we can receive information from the robot.

However, we don't have a way to represent the status of a particular robot.

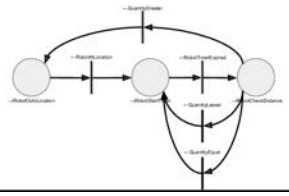
Tokens

With the addition of output events we can send commands to the robot.
With the addition of input events we can receive information from the robot.

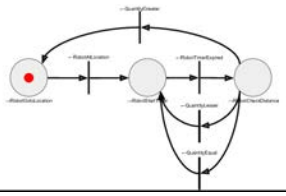
However, the current SPN doesn't have a way to represent the status of a particular robot. Is the robot going to its location, waiting on its timer, or checking its distance?



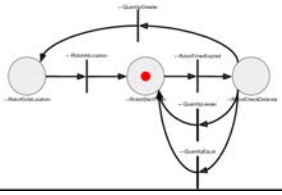
We will accomplish this using "tokens." We will put a token in a place if the robot's status corresponds to that place. Tokens only exist in places: not transitions or edges.



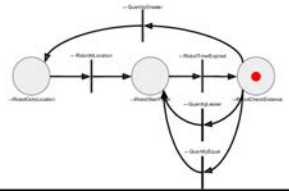
Here the token tells us that the robot is currently moving to its station keeping location.



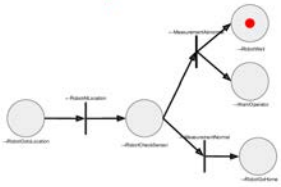
Here the token tells us that the robot is waiting.



Here the token tells us that the robot is checking its distance from the station keep location.



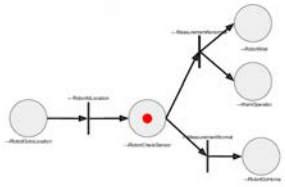
Quiz 4-1: Identify the status of the robot



Quiz 4-1 Solution

Robot is waiting

Quiz 4-2: Identify the status of the robot



Quiz 4-2 Solution

Robot is checking sensor measurement

With the addition of output events we can send commands to the robot.
With the addition of input events we can receive information from the robot.
With the addition of tokens we can represent the status of the robot.

However, we don't have a method for moving the robot's token between places as its status changes.

Edge Requirements

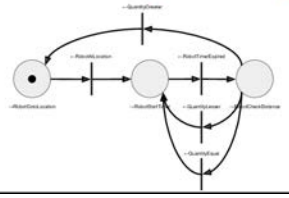
With the addition of output events we can send commands to the robot.
With the addition of input events we can receive information from the robot.
With the addition of tokens we can represent the robot's status.

However, we don't have a method for moving the robot's token between places as its status changes. We will do this by firing transitions as status changes occur and subsequently moving tokens between places according to edge requirements.

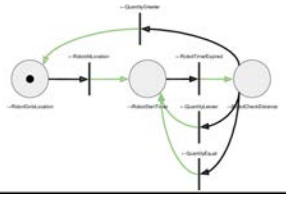
There are two types of edge requirements. First we will look at out edge requirements.

Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires

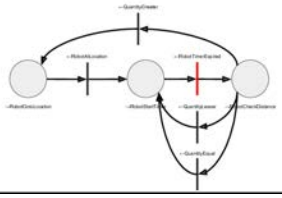
Quiz 5-1: Identify the out edges



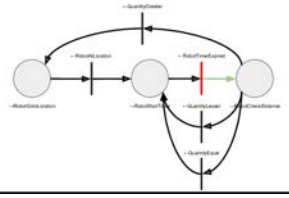
Quiz 5-1 Solution



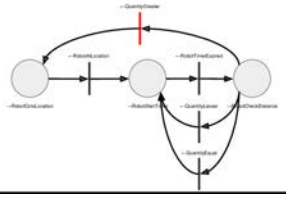
Quiz 5-2: Identify the out edges connected to the "firing" transition



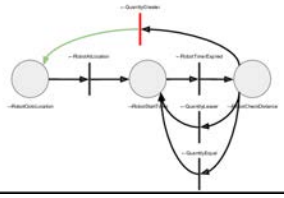
Quiz 5-2 Solution



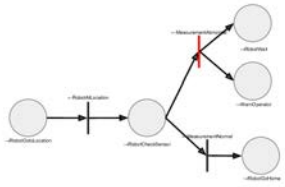
Quiz 5-3: Identify the out edges connected to the "firing" transition



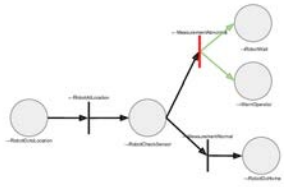
Quiz 5-3 Solution



Quiz 5-4: Identify the out edges connected to the "firing" transition



Quiz 5-4 Solution

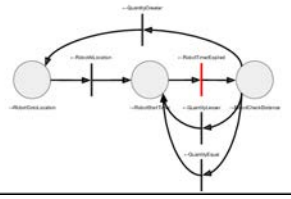


Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires

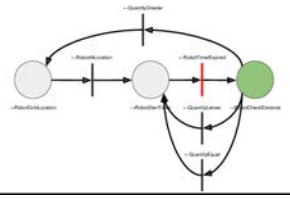
An out edge requirement can affect tokens several places: "out" places and "in" places.

- "Out" places are places connected to the firing transition via one of its out edges

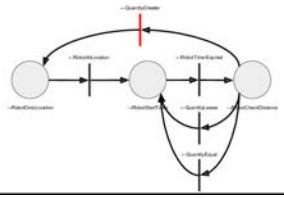
Quiz 5-5: Identify the out places for the "firing" transition



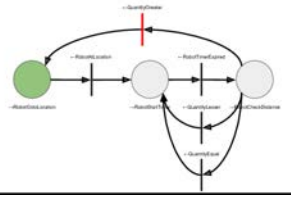
Quiz 5-5 Solution



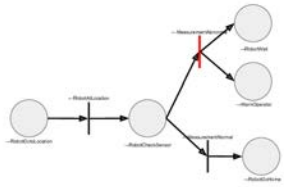
Quiz 5-6: Identify the out places for the transition



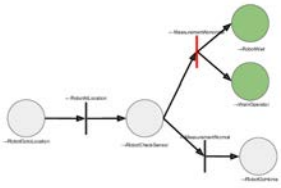
Quiz 5-6 Solution



Quiz 5-7: Identify the out places for the transition



Quiz 5-7 Solution

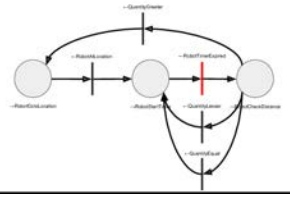


Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires.

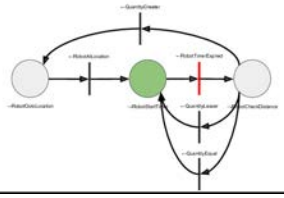
The requirements can affect tokens in certain places: "in" places and "out" places.

- "Out" places are places connected to the firing transition via an out edge
- "In" places are places connected to the firing transition via an in edge

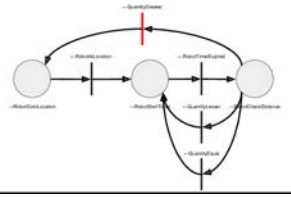
Quiz 5-8: Identify the in places for the transition



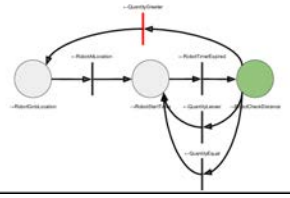
Quiz 5-8 Solution



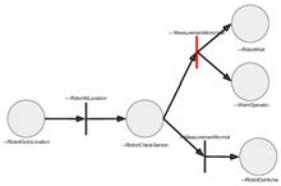
Quiz 5-9: Identify the in places for the transition



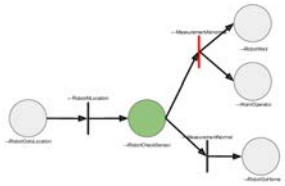
Quiz 5-9 Solution



Quiz 5-10: Identify the in places for the transition



Quiz 5-10 Solution



Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires

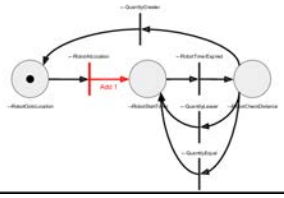
The requirements can affect tokens in certain places: "in" places and "out" places.

- "Out" places are places connected to the firing transition via an out edge
- "In" places are places connected to the firing transition via an in edge

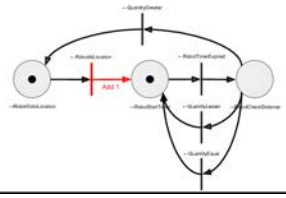
For now, let's consider 3 options for what we can do to tokens when a transition fires

- Add: Adds a specified number of tokens to all out places

Example: Here is the SPN before the transition highlighted in red fires



Example: Here is the SPN after the transition highlighted in red fires and 1 token is added.



While it is fine to have tokens in multiple places and also fine to have multiple tokens in a place, here it doesn't make much sense. We do not want the robot to be moving and waiting at the same time.

Let's look at another option.

Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires

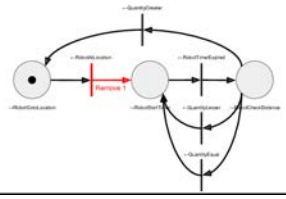
The requirements can affect tokens in certain places: "in" places and "out" places.

- "Out" places are places connected to the firing transition via an out edge
- "In" places are places connected to the firing transition via an in edge

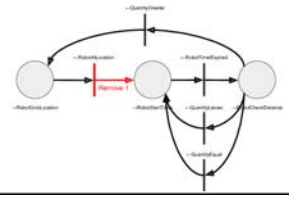
For now, let's consider 3 options for what to do with tokens

- Add: Adds a specified number of tokens to all out places
- Consume: Removes a specified number of tokens from all in places, if possible

Example: Here is the SPN before the transition highlighted in red fires



Example: Here is the SPN after the transition highlighted in red fires and removes 1 token.



Now there are no tokens in the SPN, so we do not know what the status of the robot is. Clearly this does not make sense.

Let's look at the third option.

Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires

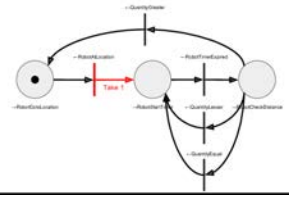
The requirements can affect tokens in certain places: "in" places and "out" places.

- "Out" places are places connected to the firing transition via an out edge
- "In" places are places connected to the firing transition via an in edge

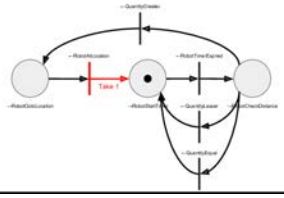
For now, let's consider 3 options for what to do with tokens

- Add: Adds a specified number of tokens to all out places
- Consume: Removes a specified number of tokens from all in places, if possible
- Take: Removes a specified number of tokens from all in places, if possible, AND adds the specified number of tokens to all out places

Example: Here is the SPN before the transition highlighted in red fires



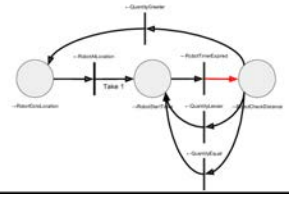
Example: Here is the SPN after the transition highlighted in red fires and takes 1 token



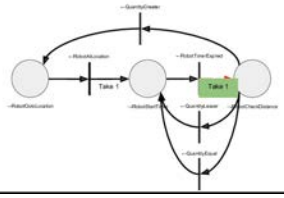
Now we have moved the token from the "Robot is moving to location" place into the "Robot is waiting" place. As we would expect, the robot is in only one of the places describing its status.

Let's fill out the rest of the out edge requirements.

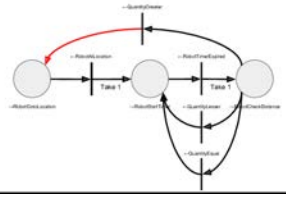
Quiz 5-11: Choose the requirement for the out edge



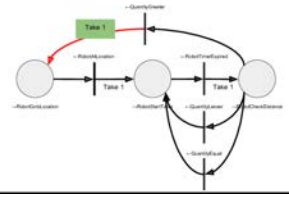
Quiz 5-11 Solution



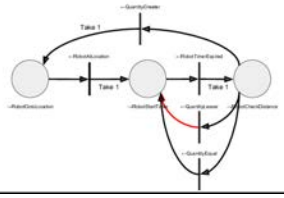
Quiz 5-12: Choose the requirement for the out edge



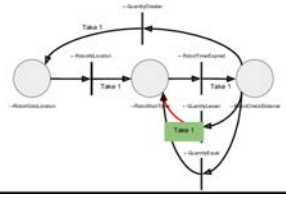
Quiz 5-12 Solution



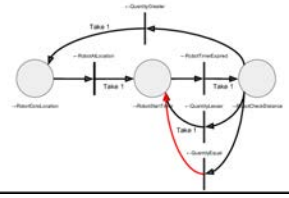
Quiz 5-13: Choose the requirement for the out edge



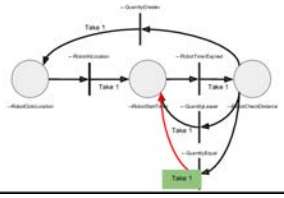
Quiz 5-13 Solution

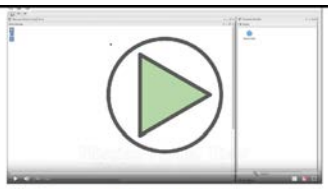


Quiz 5-14: Choose the requirement for the out edge



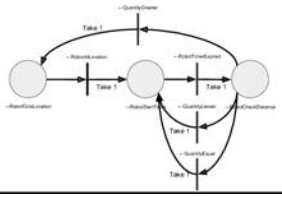
Quiz 5-14 Solution





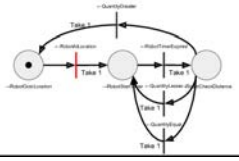
Watch "Output Requirements": This video will show you how to add out edge requirements.

Job 5-1: Add the following out edge requirements to the SPN



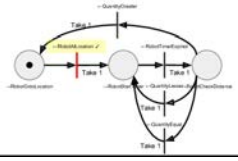
Now we know what to do with tokens when a transition fires. Next we will talk about WHEN a transition should fire: when certain information has been received and certain tokens are present.

In this scenario, when the transition with "←RobotAtLocation" fires, it will move the robot's token from place with "→RobotGotoLocation" to the place with "→RobotStartTimer." We want the transition to fire when two conditions are met:
1) Robot was moving to its location
2) Robot arrived at its destination



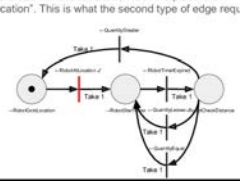
Let's look at #2 first:
2) Robot arrived at its destination

We will know this when the "←RobotAtLocation" input event is received. We will put a "✓" by an input event to indicate it has been received.



Now let's look at #1
1) Robot was moving to its location

We will know this is true if the robot's token is in the place with "→RobotGotoLocation". This is what the second type of edge requirement is used for.



There are two types of edge requirements.

- Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires
- In edge requirements: Are put on In Edges and list tokens necessary for a transition connected to it to fire

There are two types of edge requirements.

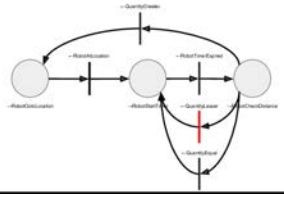
- Out edge requirements: Are put on Out Edges and describe what to do with tokens when a transition connected to it fires
- In edge requirements: Are put on In Edges and list tokens necessary for a transition connected to it to fire

Specifically, in edge requirements can require that a number of tokens be present OR be absent from the connected place in order for the transition to fire.

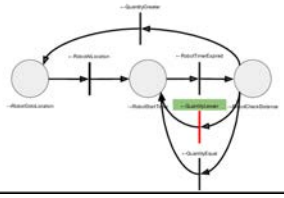
Now for a transition to fire, each of the following must be true.

- All input events on the transition must have been received
- All in edge requirements on in edges connected to the transition must be satisfied

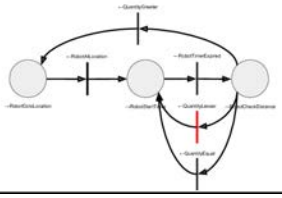
Quiz 5-15: Select the input events necessary for the transition to fire



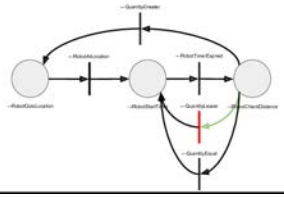
Quiz 5-15 Solution



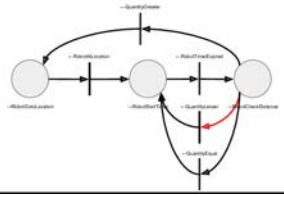
Quiz 5-16: Select the in edges connected to the transition



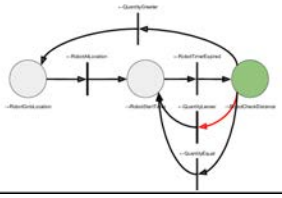
Quiz 5-16 Solution



Quiz 5-17: Select the place connected to the in edge



Quiz 5-17 Solution

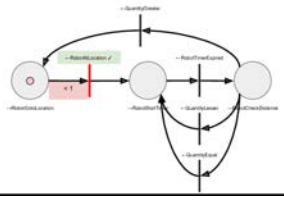


In edge requirements: Are put on In Edges and list tokens necessary for a transition connected to it to fire

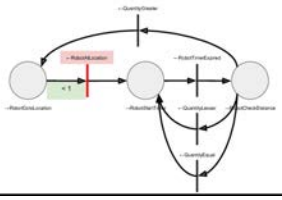
For now, let's consider 2 options for requirement criteria

- Less Than: The place connected to the in edge must have less than a specified number of tokens

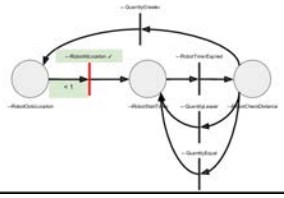
The transition does NOT fire: the input event on the transition has been received, but the in edge requirement is not satisfied



The transition does NOT fire: the in edge requirement is satisfied, but the input event on the transition has not been received



The transition does fire: the in edge requirement is satisfied and the input event on the transition has been received

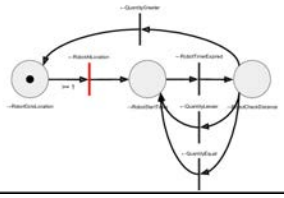


In edge requirements: Are put on In Edges and list other requirements necessary for a transition connected to it to fire

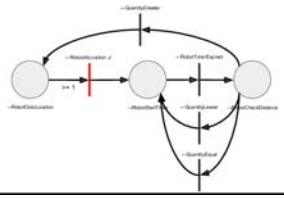
For now, let's consider 2 options for requirement criteria

- Less Than: The place connected to the in edge must have less than a specified number of tokens
- Greater Than or Equal To: The place connected to the in edge must have greater than or equal to a specified number of tokens

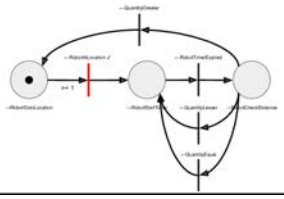
The transition does NOT fire: the in edge requirement is satisfied, but the input event on the transition has not been received.



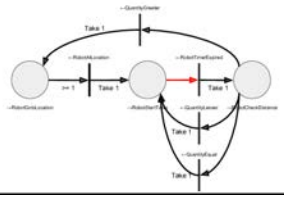
The transition does NOT fire: the input event on the transition has been received, but the in edge requirement is not satisfied



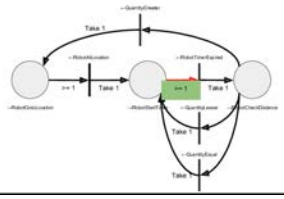
The transition does fire: the input event on the transition has been received and the in edge requirement is satisfied



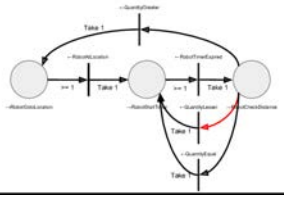
Quiz 5-18: Choose the requirement for the in edge



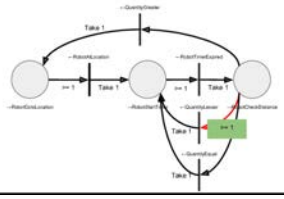
Quiz 5-18 Solution



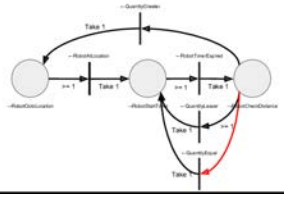
Quiz 5-19: Choose the requirement for the in edge



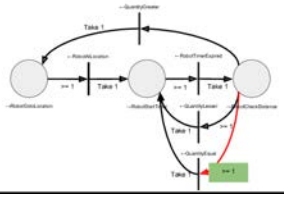
Quiz 5-19 Solution



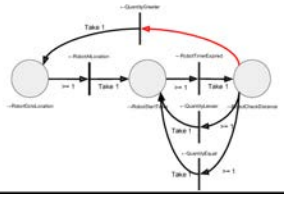
Quiz 5-20: Choose the requirement for the in edge



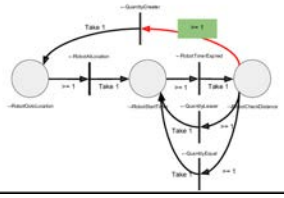
Quiz 5-20 Solution

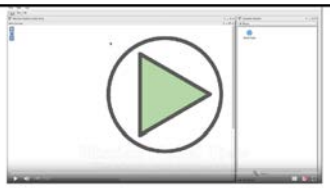


Quiz 5-21: Choose the requirement for the in edge



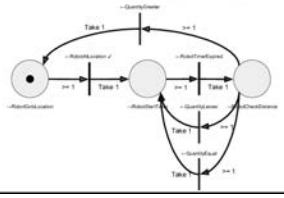
Quiz 5-21 Solution



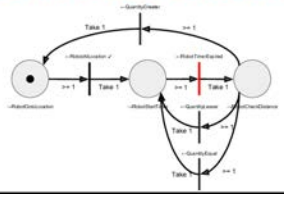


Watch "Input Requirements": This video will show you how to add in edge requirements.

Job 5-2: Add the in edge requirements



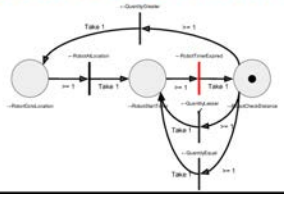
Quiz 5-22: Does the transition fire?



Quiz 5-22 Solution

No: ←RobotTimerExpired has not been received and the in edge requirement is not satisfied (→RobotStartTimer place has 0 tokens and would need at least 1 token)

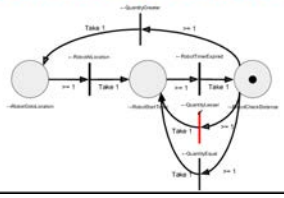
Quiz 5-23: Does the transition fire?



Quiz 5-23 Solution

No: ←RobotTimerExpired has not been received and the in edge requirement is not satisfied (→RobotStartTimer place has 0 tokens and would need at least 1 token)

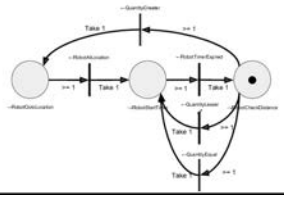
Quiz 5-24: Does the transition fire?



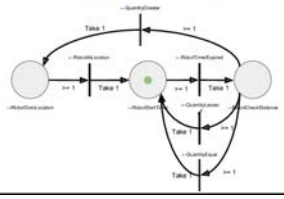
Quiz 5-24 Solution

Yes: ←QuantityLesser has been received and the in edge requirement is satisfied (→RobotCheckDistance place has one token)

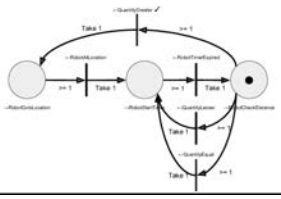
Quiz 5-25: Draw the tokens after all eligible transitions fire



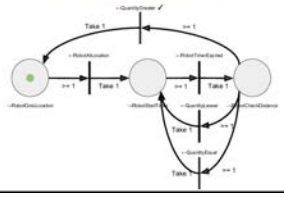
Quiz 5-25 Solution



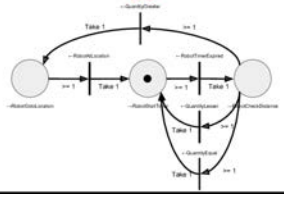
Quiz 5-26: Draw the tokens after all eligible transitions fire



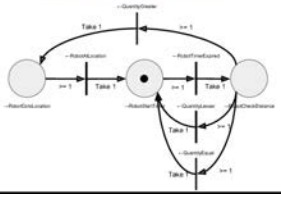
Quiz 5-26 Solution



Quiz 5-27: Draw the tokens after all eligible transitions fire



Quiz 5-27 Solution

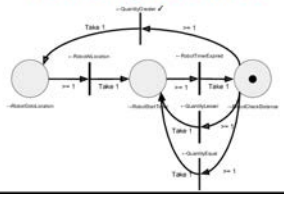


In addition to manipulating tokens, when a transition fires it also resets the receipt status of each of its input events to "not received" The transition firing means that the information received in the input event is being acted on, so the transition should not fire again in the future unless that information is received again.

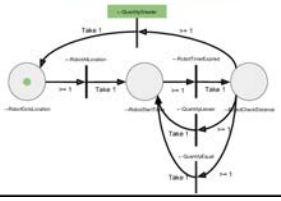
Once a firing transition manipulates tokens according to the its out edge requirements and its input events have been marked as "not received," other transitions are allowed to check if they should fire.

Note that, if an input event is received, but the transition does not fire, it remains "received" until the transition does fire.

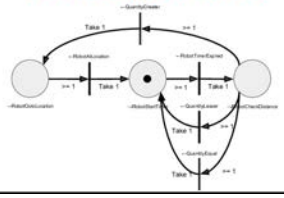
Quiz 5-28: Draw the tokens and update the input event receipt status after all eligible transitions fire



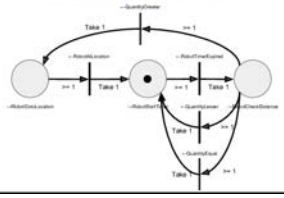
Quiz 5-28 Solution



Quiz 5-29: Draw the tokens and update the input event receipt status after all eligible transitions fire



Quiz 5-29 Solution



We have now addressed when to fire a transition and what to do with tokens when a transition fires. However, the current representation works only for a single robot.

Tokens Part 2

We have now addressed when to fire a transition and what to do with input event receipt status and tokens when a transition fires. However, the current representation works only for a single robot.

What if we want many robots station keeping? We would need to keep track of what each robot is doing individually as they will be in different locations.

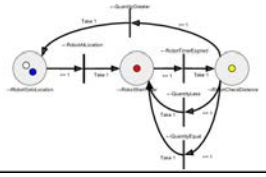
We have now addressed when to fire a transition and what to do with input event receipt status and tokens when a transition fires. However, the current representation works only for a single robot.

What if we want many robots station keeping? We would need to keep track of what each robot is doing individually as they will be in different locations.

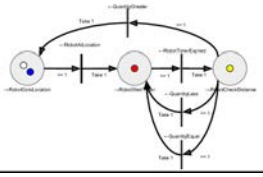
To address this we use the concept of "colored tokens," allowing tokens to optionally store information, such as the ID of a robot. To make it easy to visualize, we will assign each robot ID a color and color its token accordingly.

If a token has no robot ID attached to it, we call it a "generic token" and color it black. Generic tokens are used in parts of plans which don't involve robots.

Here we can see the station keeping plan running with tokens for White, Blue, Red, and Yellow robots. No generic tokens are used.



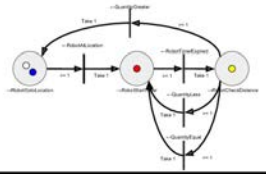
QUIZ 6-1: What is white robot doing?



Quiz 6-1 Solution

White robot is moving to the station keep location

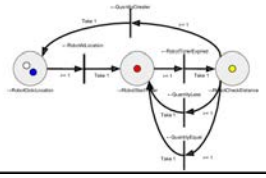
QUIZ 6-2: What is blue robot doing?



Quiz 6-2 Solution

Blue robot is moving to the station keep location

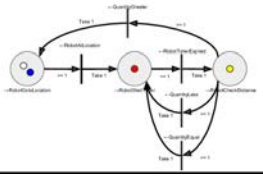
QUIZ 6-3: What is red robot doing?



Quiz 6-3 Solution

Red robot is waiting

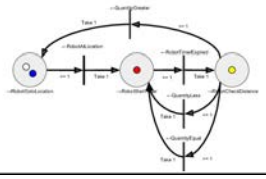
QUIZ 6-4: What is yellow robot doing?



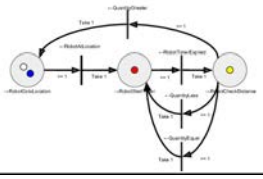
Quiz 6-4 Solution

Yellow robot is checking its distance from the station keep location

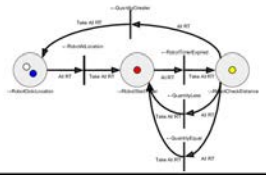
However, this representation causes a problem with the edge requirements as there is no way of distinguishing between tokens. Consider this: when Blue robot finishes moving to its location and generates a ←RobotAtLocation input event, how will we know to just move the Blue token?



We solve this with the concept of "Relevant Tokens" (RT). An input event can list any number of tokens which are relevant to their generation. When Blue robot finishes moving to its location, a ←RobotAtLocation input event will be generated with "Blue robot token" as a relevant token. We also modify the edge requirements to only manipulate the set of relevant tokens (RT).



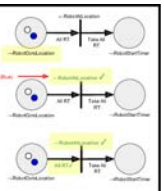
Let's work through an example looking at a small portion of the plan.



1. Blue and White robots begin moving to their locations.

2. Blue robot finishes moving and generates a ←RobotAtLocation, listing Blue robot token as a RT. We mark ←RobotAtLocation as being received.

3. All input events on the transition have been received, so we check if all its in edge requirements are satisfied. The only in edge requirement is "All Relevant Tokens," so we look at the list of relevant tokens for each input event. The only input event is ←RobotAtLocation, and its relevant token list is simply Blue robot token. Blue robot token is in "←RobotGotoLocation," so the in edge requirement is satisfied.

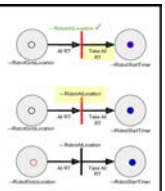


4. All input events have been received and all in edge requirements have been satisfied, so the transition fires. The out edge requirement is "Take All Relevant Tokens," and the list of relevant tokens is simply Blue robot token. Blue robot token is removed from all places connected to the transition by an in edge and added to the place connected to the out edge. So the blue token is removed from "←RobotGotoLocation" and added to "←RobotStartTimer."

5. We have manipulated the tokens, so now the input events are marked as "not received" and we are done firing the transition.

6. White robot continues moving to its location.

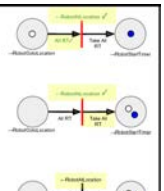
7. White robot finishes moving and generates a ←RobotAtLocation, listing White robot token as a Relevant Token. We mark ←RobotAtLocation as being received.

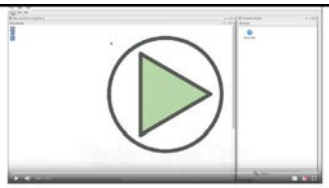


8. All input events on the transition have been received, so we check if all its in edge requirements are satisfied. The only in edge requirement is "All Relevant Tokens," so we look at the list of relevant tokens for each input event. The only input event is ←RobotAtLocation, and its relevant token list is simply White robot token. White robot token is in "←RobotGotoLocation," so the in edge requirement is satisfied.

9. All input events have been received and all in edge requirements have been satisfied, so the transition fires. The out edge requirement is "Take All Relevant Tokens," and the list of relevant tokens is simply White robot token. White robot token is removed from all places connected to the transition by an in edge and added to the place connected to the out edge. So the white token is removed from "←RobotGotoLocation" and added to "←RobotStartTimer."

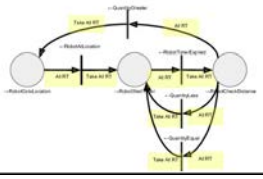
10. We have manipulated the tokens, so now the input events are marked as "not received" and we are done firing the transition.





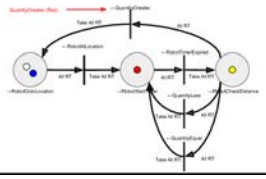
Watch "Using Relevant Tokens": This video will show you how to use new RT in and out edge requirements.

Job 6-1: Update the edge requirements to use RT



QUIZ 6-5: What would happen if the following input event was received?

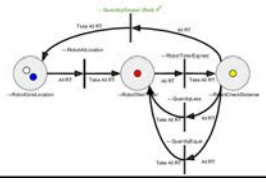
Quantity Greater, Relevant Tokens = Red robot token



Quiz 6-5 Solution Part I

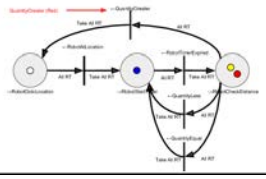
←QuantityGreater would be marked as received, but the transition would not execute because the red token is not in the place with →RobotCheckDistance

Quiz 6-5 Solution Part II



QUIZ 6-6: What would happen if the following input event was received?

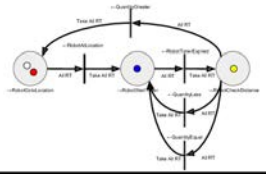
Quantity Greater, Relevant Tokens = Red robot token



Quiz 6-6 Solution Part I

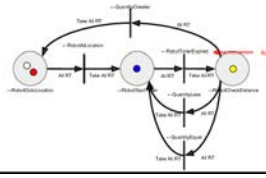
The transition with ←QuantityGreater would execute and move the red token from the place with →RobotCheckDistance to the place with →RobotGotoLocation

Quiz 6-6 Solution Part II



QUIZ 6-7: What would happen if the following input event was received?

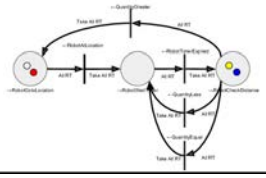
Robot Timer Expired, Relevant Tokens = Blue robot token



Quiz 6-7 Solution Part I

The transition with ←RobotTimerExpired would execute and move the blue token from the place with →RobotStartTimer to the place with →RobotCheckDistance

Quiz 6-7 Solution Part II



We have now discussed how to move specific robot tokens through the SPN, but have not talked about how the actual commands are sent to robots to make them do things.

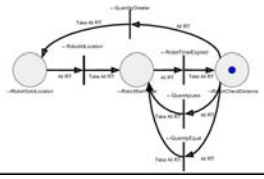
When a token or set of tokens enter a place, each output event on the place is activated by the tokens. Output events do different things depending on what tokens activate it.

Consider the output event →RobotGotoLocation:

- If activated by a robot token, the robot is told to move to the location
- If activated by a generic token, nothing is done

QUIZ 6-8: What would happen in the following scenario?

Blue robot token enters the place with "→RobotCheckDistance"

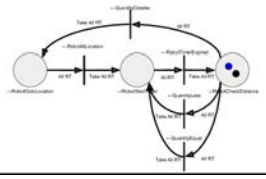


Quiz 6-8 Solution

Blue robot is commanded to check its distance from the desired location, and generate a ←QuantityGreater, ←QuantityLess, or ←QuantityEqual accordingly. The generated input event will list blue robot as a relevant token.

QUIZ 6-9: What would happen in the following scenario?

Blue robot token and 1 generic token both enter the place with "→RobotCheckDistance"



Quiz 6-9 Solution

Blue robot will be commanded as in the previous example.

The generic token will be ignored.

Now we will discuss how SPNs start and how SPNs end.

Start Place

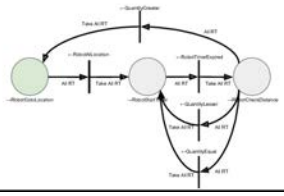
Each SPN has exactly one start place where the following tokens are placed when the SPN is started

- 1 Generic token
- 1 Robot token for each existing robot

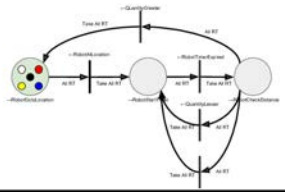
If the start place has any output events, all of the tokens activate them.

The start place is colored green.

Let's make the place with the "→RobotGotoLocation" the start place.



Now if we start the plan with White, Red, Yellow, and Blue robots connected, the following tokens are added to the start place.

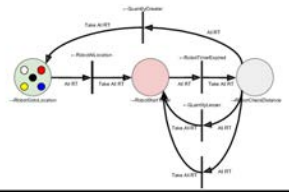


End Place

Each SPN has any number of end places which terminate all plan activity when a token enters it, such as stopping any robot movement initiated by that SPN.

End places are colored red.

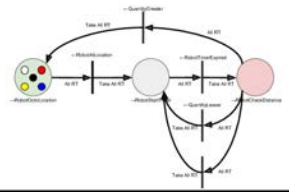
QUIZ 6-10: What will happen if we make →RobotStartTimer an End place?



Quiz 6-10 Solution

The plan will end as soon as the first robot reaches its destination.

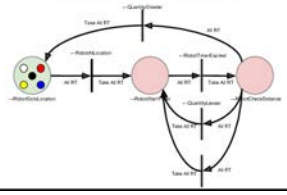
QUIZ 6-11: What will happen if we make →RobotCheckDistance an End place?



Quiz 6-11 Solution

The plan will end as soon as the first robot timer expires.

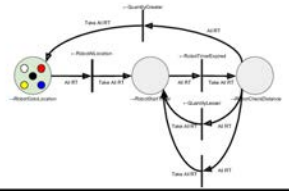
QUIZ 6-12: What will happen if we make →RobotStartTimer and →RobotCheckDistance End places?



Quiz 6-12 Solution

The plan will end as soon as the first robot reaches its destination.

Currently there is no place which makes a good end place. Next we will talk about interacting with the operator for information, such as when to end a SPN.



Operator Interaction

Right now there is no end place for our station keeping plan. Let's add an end place that is triggered by the operator.

We'll do this with the "→OperatorApprove" output event. When activated, the operator will see a "Yes / No" dialog box on the GUI.

- If the operator clicks Yes, the dialog box closes and an "←OperatorApproved" input event is generated
- If the operator clicks No, the dialog box closes and an "←OperatorRejected" input event is generated

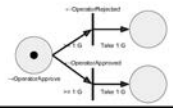
Note that this interaction does not depend on any Robots, which means two things:

- The generated input event will not have any Relevant Tokens
- We can use a Generic token to activate the →OperatorApprove output event

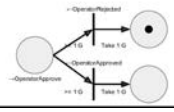
A Robot token can also be used to activate →OperatorApprove, but the generated ←OperatorApproved or ←OperatorRejected input event still will not have any Relevant Tokens.

Remember that the inverse is not true: an output event that uses robots (→RobotGotoLocation) can only be activated by Robot tokens and not Generic tokens, while output events that don't use robots can be activated by Robot or Generic tokens.

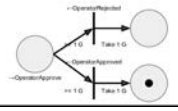
Below is an example sequence activating a →OperatorApprove with a Generic token. Note that instead of ">= 1" and "Take 1" as in previous lessons, we now specify the type of token, such as Generic (G), Robot, or Relevant Token (RT).



Here is what the SPN would look like if the the operator chooses "No"

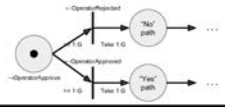


Here is what the SPN would look like if the the operator chooses "Yes"



Notice that there are two different paths: the "No" path and the "Yes" path, which split out of the "→OperatorApprove" place. This is called branching.

In this case, we will only enter one of the two branches, depending on what the operator chooses.



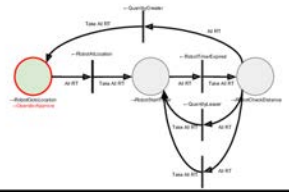
Now we need to decide how to use the sequence in our station keep SPN.

We can use →OperatorApprove to ask the operator to approve ending the SPN

- If they answer yes, the plan should end
- If they answer no, the operator should be asked again later

Let's look at adding the →OperatorApprove output event to an existing place in the SPN and consider when the dialog box will pop up. Remember that we can have multiple output events on a place and multiple input events on a transition.

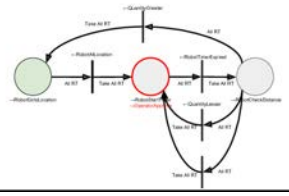
QUIZ 7-1: When will the dialog box appear if we add →OperatorApprove to the list of output events on the first place?



Quiz 7-1 Solution

Every time any robot is told to move to the station keep location.

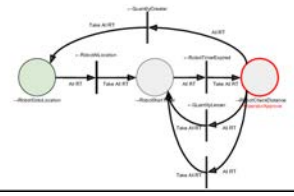
QUIZ 7-2: When will the dialog box appear if we add →OperatorApprove to the list of output events on the second place?



Quiz 7-2 Solution

Every time any robot reaches the station keep location/a robot starts waiting.

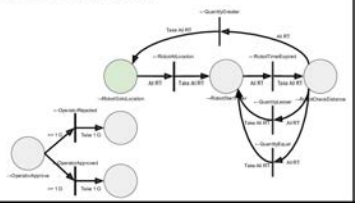
QUIZ 7-3: When will the dialog box appear if we add →OperatorApprove to the list of output events on the third place?



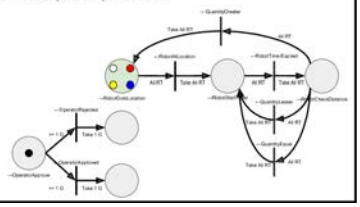
Quiz 7-3 Solution

Every time any robot finishes waiting/checks its distance to the station keep location.

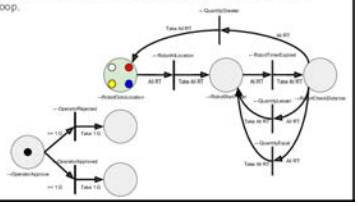
To avoid the dialog box from popping up continuously, we need the operator decision to be outside of the station keep loop.



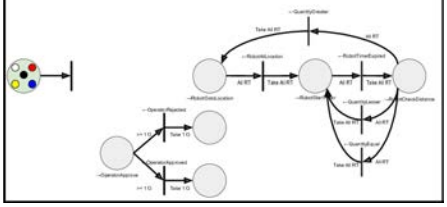
If we use the "branching" concept, we can separate the operator question from the station keep movement. When the plan starts, we will want the robot tokens put in the station keep branch and a generic token put in the question branch.



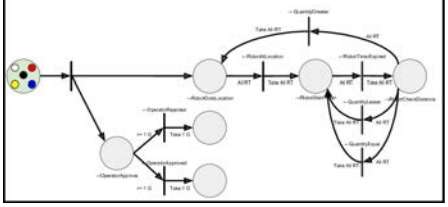
We can only have one start place, which is inside the the station keeping loop. We need to move the start place out of the loop to avoid it getting triggered each station keep loop.



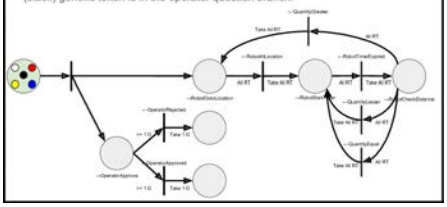
Here we add the new start place.



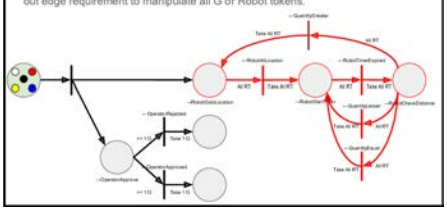
Now we can create the branches to the station keep and operator question.



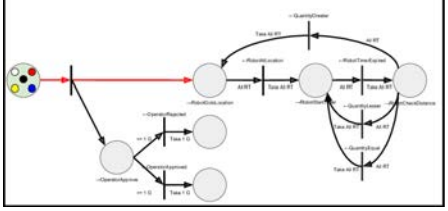
Now we need to figure out the edge requirements for this new section so that the White, Red, Yellow, and Blue robot tokens are in the station keep branch and the (black) generic token is in the operator question branch.



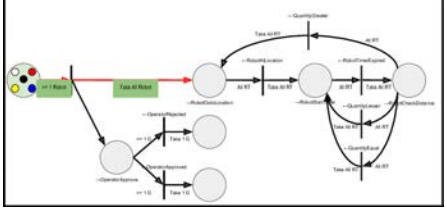
We need the robot tokens in this part of the plan. Similar to the previous lesson where an out edge requirement could manipulate all RT, we can also instruct an out edge requirement to manipulate all G or Robot tokens.



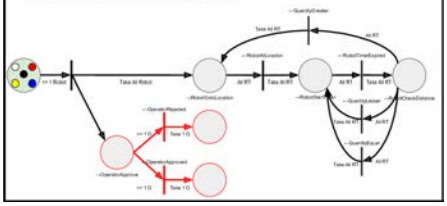
Job 7-1: Add the edge requirements.



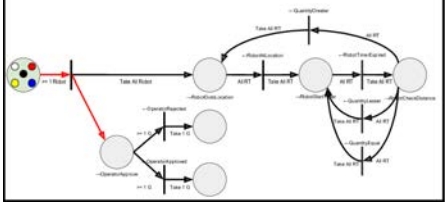
Job 7-1 Solution



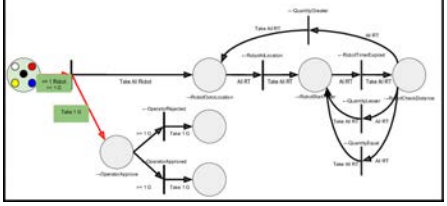
We need a generic token in this part of the plan. Remember that multiple requirements can be listed on a single edge.



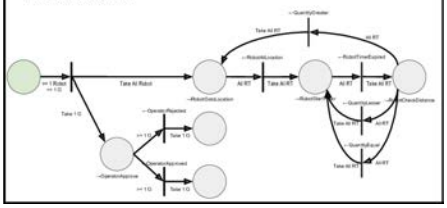
Job 7-2: Add the edge requirements.



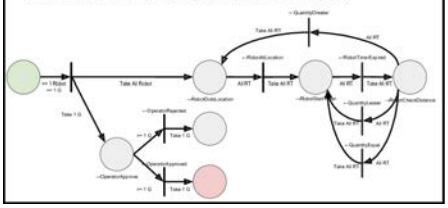
Job 7-2 Solution



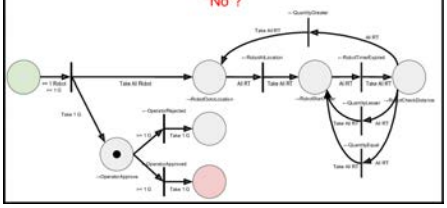
Now that we have the edge labels, let's customize what happens depending on the operator's decision.



If the operator chooses "Yes", we want the plan to end. So we should make the place connected to the OperatorRejected input event an end place.



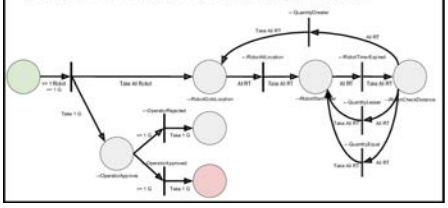
QUIZ 7-4: What will happen in this case if the operator clicks "No"?



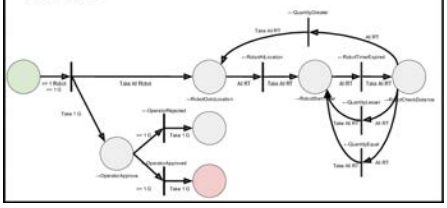
Quiz 7-4 Solution

The generic token will be moved to the place with ←OperatorRejected and the question will never be asked again

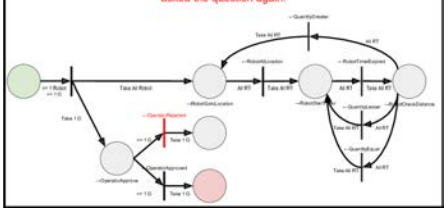
If the operator clicks "No," the plan continues as normal, but the question is never asked again. This means the operator cannot end the plan in the future.



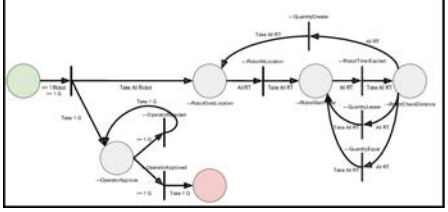
Let's change it so that if the operator accidentally presses "No" they are asked the question again.



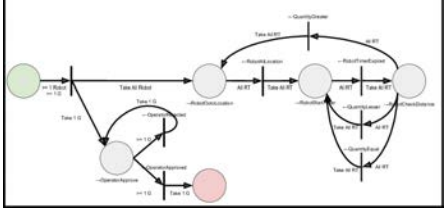
Job 7-3: Change the SPN so that if the operator accidentally presses "No," they are asked the question again.



Job 7-3 Solution



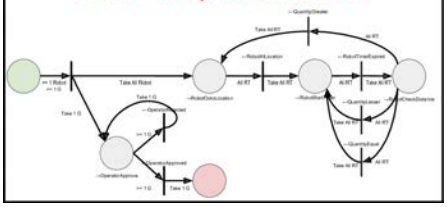
Now if the operator accidentally clicks "No," the questions will pop back up again.



Watch "Setting a Start Place". This video will show you how to set a place as a start place.

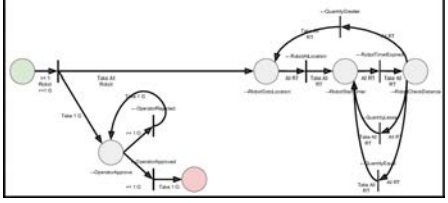
Watch "Setting an End Place". This video will show you how to set a place as an end place.

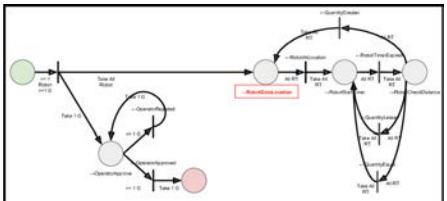
Job 7-4: Update the SPN



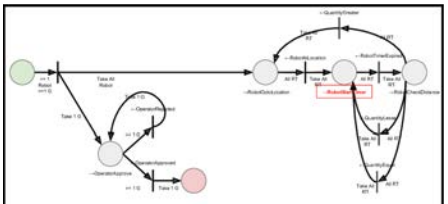
Variables

Let's consider the pieces of information needed to run the station keep SPN:

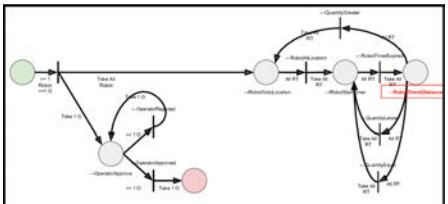




Where should the boats go?



How long should the timer last?



What location should the boat calculate its distance to?
How far is "too far"?

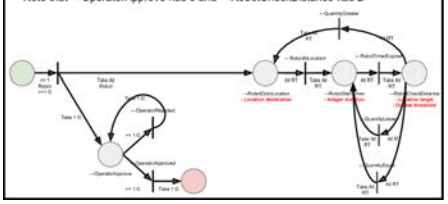
Each of these pieces of information is called a "field." Each output and input event can have any number of fields. Each field has a type, name, and assignable value.

For output events, the values of their fields are used to make the request to the operator, robot, or AI.

For input events, the values of their fields are assigned by the operator, robot, or AI when the input event is generated.

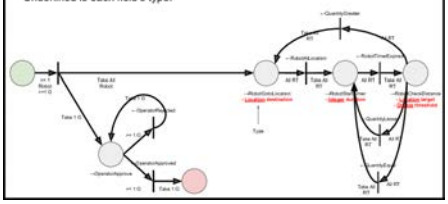
First let's look at each output event's fields.

Note that →OperatorApprove has 0 and →RobotCheckDistance has 2



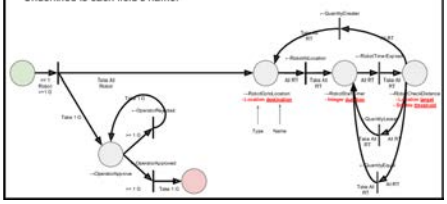
First let's look at each output event's fields.

Underlined is each field's type.



First let's look at each output event's fields.

Underlined is each field's name.



For output event fields, there are 3 ways to assign it a value (also called a definition).

1- Assign it a value when developing the SPN

If we know the value we want to use when developing the SPN in DREAMM, we can assign it then.

For example, let's say we always want to wait 10 seconds between distance checks.

Watch "Setting a Static Value": This video will show you how to define the value of an output event's field in DREAMM.

Job 8-1: Define the →RobotStartTimer output event's Integer duration to 10 seconds

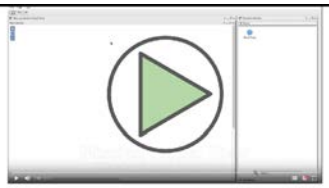
For output events, there are 3 ways to assign a value (also called a definition) to a field.

1- Assign it a value when developing the SPN

2- Assign it a value when starting the SPN

If we won't know the value of a field until the the operator starts the plan, we can have the operator define the value at that point.

For example, in most cases the station keep location will not be known ahead of time and the operator will need to specify the location when the SPN is started.



Watch "Using a Runtime Value": This video will show you how to have the value of an output event's field be filled out at runtime.

Job 8-2: Verify →RobotGotoLocation output event's location field is not defined

For output events, there are 3 ways to assign a value (also called a definition) to a field.

1- Assign it a value when developing the SPN

2- Assign it a value when starting the SPN

3- Use the value in another input or output event's field

If we want an output event's field to use the same value another event's field is using, we use a variable.

A variable consists of a type, name, and a value, similar to an event's field. Initially a variable has no value.

Variables can either be written to or read from:

- Write: Set the value of the variable
- Read: Get the value of the variable

For each field whose value we want to use elsewhere in the plan, we tell it to write to a variable name.

For each output event field we want to use that written value, we tell it to read from that variable name.

There are 2 situations where we would tell an output event's field to write to a variable:

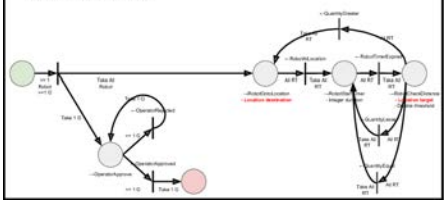
1- A field's value is assigned when the SPN is developed and we want to use that same value in other fields

We could assign the desired value to each of these fields, but if we decide to change the value later, we have to change the value for each field.

Instead, we can assign the value to just one of these fields and also tell it to write to a variable. Then we tell each of the other fields to read from that variable name.

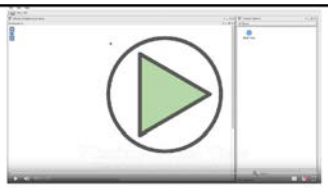
This way, if we want to change the value in the future, we just have to change it in one place.

In our station keeping SPN, the operator has to specify the station keeping location twice: once for →RobotGotoPoint, and a second time for →RobotCompareDistance.



Consider a scenario where the station keep location is known when the plan is developed. To have the two use the same location, we would do the following

1- Define the value in the →RobotGotoLocation output event's Location destination field
2- Specify a variable name for that value to be written to
3- Have the →RobotCheckDistance output event's Location target read from that variable



Watch "Saving a Static Value": This video will show you how to define the value of an output event's field and save it to a variable in DREAMM (steps 1 and 2, but not 3).

Job 8-3: Define the →RobotGotoLocation output event's Location destination field and have its value be saved to the variable "destination"

There are 2 situations where we would tell an output event's field to write to a variable:

1- The field's value is assigned when the SPN is developed and we want to also use that value elsewhere

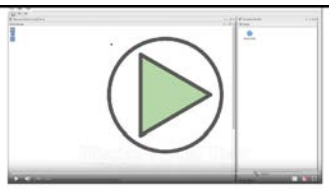
2- The field's value is assigned when the SPN is started and we want to also use that value elsewhere

We could have the operator assign the value for each of these fields when the SPN is started, but they may accidentally assign two different values.

Instead, we can tell one of these fields to write the value it will receive to a variable and tell the other fields to read from that variable.

Consider a scenario where the station keep location is known when the plan is developed. To have the two use the same location, we would do the following

1- Leave the value in the →RobotGotoLocation output event's Location destination field blank so it is defined at startup
2- Specify a variable name for the value to be written to when it is received
3- Have the →RobotCheckDistance output event's Location target read from that variable



Watch "Saving a Runtime Value". This video will show you how to save an output event field which will be defined at startup to a variable in DREAMM (steps 1 and 2, but not 3).

Job 8-4: Make the →RobotGotoLocation output event's Location destination field be defined at startup and have its value be saved to the variable "destination"

Currently we tell each robot to move to the same location, so they will constantly hit each other trying to move to the same spot. We can use an AI service to compute a unique spot near a provided location for each robot, then tell each robot to move to their unique location to avoid collisions.

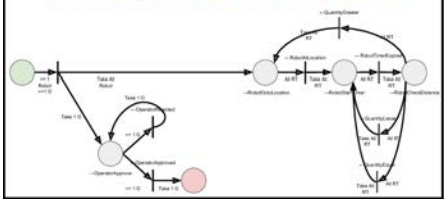
AI services use a library of algorithms to perform computations that would be difficult and/or time consuming for a human operator to perform at run time.

To have an AI to compute a unique spot, we use the following events:

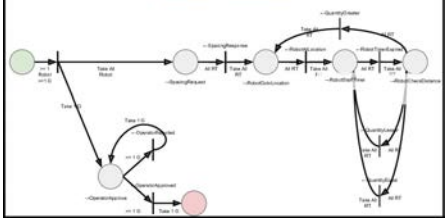
→SpacingRequest (Output event): Tells the AI to compute a unique location near a specified location for each robot token used to activate the event

←SpacingResponse (Input event): Returns the unique locations for the robots

QUIZ 8-1: Create and/or select the place and transition for →SpacingRequest and ←SpacingResponse



Quiz 8-1 Solution

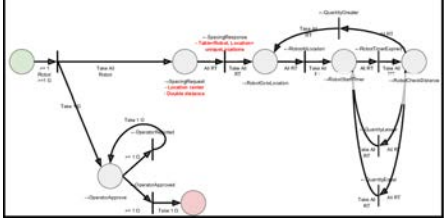


Let's look at the fields for the new events:

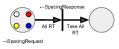
→SpacingRequest (Output event): Tells the AI service to compute a unique location near a specified location for each robot token used to activate the event
- Location center: The location that the robots' unique locations should be centered around
- Double distance: The minimum distance between any two robots' unique locations (in meters)

←SpacingResponse (Input event): Returns the unique locations for the robots
- Table<Robot, Location> uniqueLocations: A table of values: for each robot used to activate the SpacingRequest, the table has a row for that robot and its unique location.

←SpacingResponse lists the robot tokens used to activate the →SpacingRequest as Relevant Tokens.



QUIZ 8-2: What are the relevant tokens?



- Red token is Red robot's token
- Yellow token is Yellow robot's token
- Blue token is Blue robot's token
- White token is White robot's token

Quiz 8-2 Solution

The relevant tokens are
- Red robot's token
- Yellow robot's token
- Blue robot's token
- White robot's token

QUIZ 8-3: What are the relevant tokens?

- Red token is Red robot's token
- Yellow token is Yellow robot's token
- Blue token is Blue robot's token
- White token is White robot's token
- Black token is a generic token

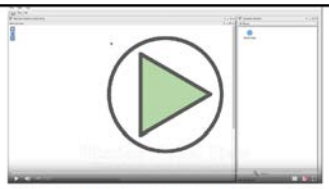
Quiz 8-3 Solution

The relevant tokens are the same as before
- Red robot's token
- Yellow robot's token
- Blue robot's token
- White robot's token

The generic token is ignored by **→SpacingRequest**

Recall that, for input events, fields are pieces of information whose value will be provided by the operator, robot, or AI when the input event is generated. ←SpacingResponse has a field "Table<Robot, Location" which we can write to a variable.

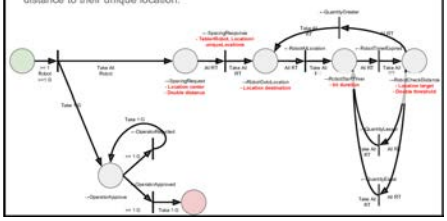
For each field in an input event, we can specify a variable name it should write its value to. Any output event fields which want to use this value can then reference the variable.



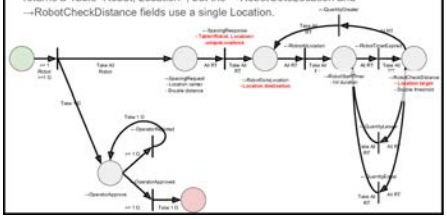
Watch "Saving an Input Event Value". This video will show you how to specify the variable an input event's field's value should be written to.

Job 8-5: Make the ←SpacingResponse input event's Table<Robot, Location uniqueLocations field write its value to the variable "spacedLocations"

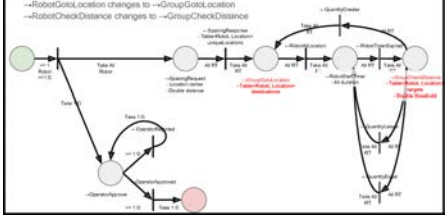
Now we want to have the proxies move to their unique location and compute their distance to their unique location.



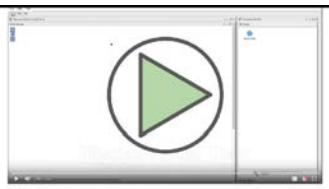
However, these types do not match. The ←SpacingReponse uniqueLocations field returns a Table<Robot, Location>, but the →RobotGotoLocation and →RobotCheckDistance fields use a single Location.



We solve this by changing output events which use a single Location to versions which instead uses a Table<Robot, Location>.
→RobotGotoLocation changes to →GroupGotoLocation
→RobotCheckDistance changes to →GroupCheckDistance



Now we can have the →GroupGotoLocation destinations field and →GroupCompareDistance targets field read from the variable we have ←SpacingResponse uniqueLocations write to.



Watch "Reading a Value from a Variable". This video will show you how to have an output event field use the value from an variable.

Job 8-6: Have the →GroupGotoLocation destinations field and →GroupCompareDistance targets field read the value from the variable ←SpacingResponse uniqueLocations writes to.

Tasks

Tasks are specific goals which we want to assign to a robot to complete. Tasks can require certain capabilities to be completed, such as a specific sensor. Tasks can also have information, such as the location where it should be performed. Here are some examples of some classes of tasks, or Task Classes:

- A "temperature" task requires a temperature sensor capability and has a location value where the temperature related task should be taken
- A "camera" task requires a camera capability and has a location value describing where the camera image should be taken from

Similarly, each robot has a list of capabilities. Robots may have different capabilities depending on their sensor payload and other hardware.

Task allocation is the process of assigning a set of tasks to a set of robots. Ideally, all task capabilities are satisfied, and the tasks will be performed in an efficient manner. For efficient execution, task allocation may consider other information, such as:

- The closest robot to the task may already be assigned to a task
- Some tasks are more important than other tasks

Here is an example of allocating 3 camera tasks and 1 temperature task to a team of 3 robots. Each robot's capabilities are listed next to its name.

Here we have allocated a task to each robot, leaving one task unallocated as there are no more robots. As these robots finish their tasks, we can repeat this process to try and allocate Camera task 3.

Consider what will happen if Robot3 finishes its task, Temperature task 1. Robot 1 and 2 are still working on their previously assigned task. We will attempt to allocate Camera task 3.

The only free robot is Robot 3, but it does not have a camera, so it cannot be allocated to Camera task 3.

Now Robot2 has finished its task, Camera task 2. Robot1 is working on its previously assigned task and Robot3 is doing nothing. We will attempt to allocate Camera task 3.

Robot2 is available and has a camera, so it is assigned the task.

Similar to how we use tokens to represent specific robots, we will also use tokens to represent specific tasks, such as Camera task 3.

When a task token is used to activate an output event, such as →GroupGotoLocation, the SPN looks up which robot is allocated to the task and sends the command to that robot.

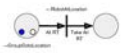
In the previous example where Camera task 3 is allocated to Robot2, when the Camera task 3 token is moved into the place with →GroupGotoLocation, Robot2 will receive that command.

We will also use the relevant token concept to move task tokens around in the same way we move robot tokens around.

When the robot finishes moving to the location, it will generate a ←RobotAtLocation input event as usual, but will list the task token as the relevant token instead of its robot token.

In the previous example, when Robot2 finishes moving to the location it was told to move to for Camera task 3, it will generate a ←RobotAtLocation input event with Camera task 3 as the relevant token.

QUIZ 9-1: What are the relevant tokens when Blue robot arrives at its location?

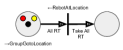


- Blue token is Blue robot's token
- White token is White robot's token

Quiz 9-1 Solution

The relevant tokens are:
- Blue robot's token

QUIZ 9-2: What are the relevant tokens when Yellow robot arrives at its location?



- Red token is a Temperature task 1's token, which is assigned to Red Robot
- Yellow token is a Camera task 1's token, which is assigned to Yellow Robot
- Black token is a generic token

Quiz 9-2 Solution

The relevant tokens are:
- Camera task 1's token

QUIZ 9-3: What are the relevant tokens when Red robot arrives at its location?

- Red token is a Temperature task 1's token, which is assigned to Red Robot
- Yellow token is a Camera task 1's token, which is assigned to Yellow Robot
- Blue token is Blue robot's token
- White token is White robot's token
- Black token is a generic token

Quiz 9-3 Solution

The relevant tokens are:
- Temperature task 1's token

Let's design a new plan using tasks to have a team of robotic boats take some both temperature measurements and panorama images at several locations in a pond. Here is some information about the team of boats:

- Some boats only have a temperature sensor
- Some boats only have a camera

This means the locations will sometimes need to be visited twice, once by a boat with a temperature sensors, and a second time with a boat with a camera.

In the end, we want the operator to

- Select the locations for temperature measurements and panoramas
- Select which computed task allocations to use

This is a lot of information to handle, so we have some new output events to help us out. Below is a list of the output events we will cover in this lesson:

→OperatorSpecifyLocations: Tells the operator to specify a list of locations on the GUI's map.

→GenerateTasks: Tells the system to create tasks of a certain type at specific locations in the world.

→TaskAllocationRequest: Tells the AI to generate task allocations for a list of Tasks

→OperatorChooseAllocation: Tells the operator to select from a list of task allocation choices or reject them all

→ApplyAllocation: Tells the system to apply a task allocation

→MeasureTemperature: Tells the robot to measure temperature at its current location, if possible

→TakePanorama: Tells the robot to take a panorama at its current location, if possible

→TaskComplete: Tells the system that a Robot has finished its assigned task

---

We also have some new input events we will cover this lesson:

←OperatorSpecifiedLocations: Returns the list of locations the operator choose in response to →OperatorSpecifyLocations

←TokensReceived: Returns a set of tokens representing tasks which were created in response to →GenerateTasks

←TaskAllocationResponse: Returns a number of task allocations in response to →TaskAllocationRequest

←TaskAllocationFailure: Indicates no task allocation was possible in response to →TaskAllocationRequest

←AllocationAccepted: Returns the task allocation the operator selected in response to OperatorChooseAllocation

←AllocationsRejected: Indicates all task allocation options were rejected by the operator in response to →OperatorChooseAllocation

←TaskAssigned: Indicates a specific task was assigned as a result of →ApplyAllocation

←TaskUnassigned: Indicates a specific task was not assigned as a result of →ApplyAllocation

←MeasurementDone: Indicates the robot is done taking a measurement as a result of →MeasureTemperature or →TakePanorama

←TaskCompleted: Indicates a task was marked as being complete via →TaskComplete

---

Here is a table showing the relationship between these new events

Output Event	Input Event(s) Generated
→OperatorSpecifyLocations	←OperatorSpecifiedLocations
→GenerateTasks	←TokensReceived
→TaskAllocationRequest	←TaskAllocationResponse or ←TaskAllocationFailure
→OperatorChooseAllocation	←AllocationAccepted or ←AllocationsRejected
→ApplyAllocation	←TaskAssigned and/or ←TaskUnassigned
→MeasureTemperature	←MeasurementDone
→TakePanorama	←MeasurementDone
→TaskComplete	

As a reminder each of these output and input events will have some number of fields. Each field has a type, name, and assignable value.

For output events, the values of their fields are used to make the request to the operator, robot, or AI.

For input events, the values of their fields are assigned by the operator, robot, or AI when the input event is generated.

Let's begin constructing this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose a task allocation
5- Apply the chosen task allocation
6- Perform and complete the assigned tasks
7- End the plan when all tasks have been completed

Let's construct this plan piece by piece:
1- Get the locations

Output events to use:
→OperatorSpecifyLocations: Tells the operator to specify a list of locations on the GUI's map.
- No fields

Input events to use:
←OperatorSpecifiedLocations: Returns the list of locations the operator choose in response to →OperatorSpecifyLocations
- Field 1:
  - Name: List of locations.
  - Type: List of Locations
  - Value: These are the locations provided by the operator
- No Relevant Tokens

Here we request and receive the list of locations from the operator.

Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens

Output events to use:
→GenerateTasks: Tells the system to create tasks at specific locations in the world.
- Field 1
  - Name: Class of tasks to create?
  - Type: Task Class
  - Value: The type of task to generate at each location, such as Temperature or Camera
- Field 2
  - Name: Locations to generate tasks from?
  - Type: List of Locations
  - Value: A list of locations where tasks should be performed. We will use the list returned by ←OperatorSpecifiedLocations.

Input events to use:
←TokensReceived: Returns a set of task tokens representing the tasks created in response to →GenerateTasks: one task token of the provided task class for each location in the provided list
- No fields
- Relevant Tokens: The Task tokens that were created

Below is the section of a SPN where task tokens are created for one task class (temperature or camera).

We add the output and input events again, this time for the other task class. Now if we activate the two →GenerateTasks, using a list of 3 locations specified by the operator, 2 ←TokensReceived input events will be generated. The first will have 3 relevant tokens: Camera task 1, Camera task 2, and Camera task 3. The second will also have 3 relevant tokens: Temperature task 1, Temperature task 2, and Temperature task 3

Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options

Output events to use:
→TaskAllocationRequest: Tells the AI to generate task allocations for a list of Tasks
- No fields

Input events to use:
←TaskAllocationResponse: Returns a number of task allocations in response to →TaskAllocationRequest
- Field 1
  - Name: Returned task allocation options.
  - Type: List of task allocations
  - Value: Each item in the list is a potential allocation
- Relevant Tokens: The Task tokens for the tasks considered in the task allocation request

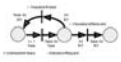
←TaskAllocationFailure: Indicates no task allocation was possible in response to →TaskAllocationRequest
- No fields
- Relevant Tokens: The Task tokens for the tasks considered in the task allocation request

---

Here we start with all unallocated task tokens in the "**// Unallocated tasks**" place. The "//" just means that "Unallocated tasks" is simply a label on the place and not an output event.

If there are task tokens which are in the "**// Unallocated tasks**" they are moved to the place with **→AllocationRequest**

The result of **→AllocationRequest** is either **←AllocationFailed** or **←AllocationsReceived**. If the system fails to compute an allocation for a set of task tokens, we it to try again. If the system succeeds in computing some task allocations for some task tokens, we want to move to the next step.



Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose an allocation

Output events to use:
→OperatorChooseAllocation: Tells the operator to select a task allocation choice or reject them all
- Field 1
  - Name: Resource allocation options to show to operator?
  - Type: List of Task Allocations
  - Value: The list of task allocation options the operator gets to choose from. We will use the list of task allocations computed by ←AllocationsReceived

---

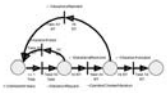
Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose an allocation

Input events to use:
←AllocationAccepted: Returns the task allocation the operator selected in response to →OperatorChooseAllocation
- Field 1
  - Name: Accepted resource allocation.
  - Type: Task Allocation
  - Value: The allocation selected by the operator
- Relevant Tokens: The Task tokens for the tasks considered in the task allocation

←AllocationsRejected: Indicates all task allocation options were rejected by the operator in response to →OperatorChooseAllocation
- No fields
- Relevant Tokens: The Task tokens for the tasks considered in the rejected task allocations

---

If the operator rejects the task allocations, we want to return them to the "**// Unallocated tasks**" place, so we can attempt to allocate them again later. If the operator accepts a task allocation, we want to move forward in the SPN.



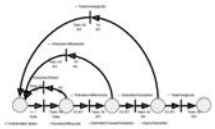
Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose an allocation
5- Apply the chosen allocation

Output events to use:
→ApplyAllocation: Tells the system to apply a task allocation
- Field 1
  - Name: Resource allocation to apply?
  - Type: Task Allocation
  - Value: The task allocation which should be applied to the system. We will use the task allocation accepted by the operator.

Input events to use:
←TaskAssigned: Indicates a task was assigned as a result of →ApplyAllocation
- No fields
- Relevant Tokens: The Task token for the task which was assigned

←TaskUnassigned: Indicates a task was not assigned as a result of →ApplyAllocation
- No fields
- Relevant Tokens: The Task token for the task which was unassigned

---

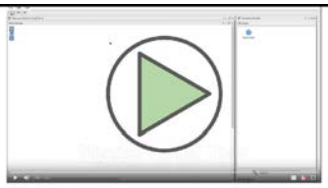


When we apply an allocation, some tasks will be allocated to a robot, while other may not, similar to our earlier example where we tried to allocate 4 tasks to 3 robots. For tasks that do not get allocated, we want to return those task tokens to the "**// Unallocated tasks**" place. For tasks that do get allocated, we want to move them forward in the SPN.

Let's construct this plan piece by piece:
1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose an allocation
5- Apply the chosen allocation
6- Perform and complete the assigned tasks

The Camera and Temperature tasks will involve taking different actions, so we need to separate the Camera Task tokens from the Temperature Task tokens. We will do this using some new edge requirement capabilities:
- We can use a new in edge requirement to require a certain number of Task tokens of a certain Task Class (Camera, Temperature, etc) be present or absent
- We can use a new out edge requirement to add, take, or consume Task tokens of a certain Task Class

For the in and out edge requirements, we have shortened "Temperature" to "Temp"



Watch "Using Task Class Requirements": This video will show you how to use in and out edge requirements which depend on Task tokens of a specific type

Now that we can split up the Task tokens by their type, we can have each task perform its specific responsibilities

For the camera task, we want the allocated robot to move to the location of the task, take the panorama, and then mark the task as complete.

For the temperature task, we want the allocated robot to move to the location of the task, measure the water temperature, and then mark the task as complete.

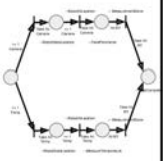
We already know how to move robots using →RobotGotoLocation and ←RobotAtLocation.

Here are additional output events to use:
→MeasureTemperature: Tells the robot to measure the water temperature at its current location

→TakePanorama: Tells the robot to take a panorama at its current location

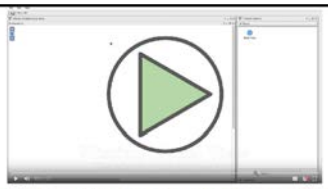
→TaskComplete: Tells the system that a Robot has finished its assigned task
- No fields

Additional input events to use:
←MeasurementDone: Indicates the robot is done taking a measurement as a result of →MeasureTemperature or →TakePanorama
- No fields
- Relevant Tokens: If →MeasureTemperature/→TakePanorama was activated with a Task token the RT is the task token; if it was activated by a Robot token the RT is the Robot token



When a Task token assigned to a robot activates the →RobotGotoLocation event, we have the problem of knowing where to send the robot. The task has an associated location, but the SPN needs to know to grab that location from the task.

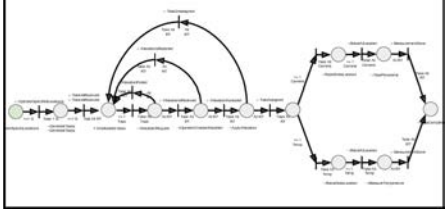
To do this, for output events which are activated by a task token and have a Location field, we can list the task as the variable to read from for the Location field. This tells the system to use the Location associated with the task for the Location field.



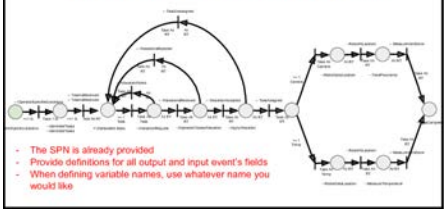
Watch "Using Task Definition for Field": This video will show you how to have →RobotGotoLocation's Location destination field use the Location associated with the Task token that activates it

Note that if a Robot token, not a Task token, activates a →RobotGotoLocation with the Location field set to use "task" there will be no destination, so it will immediately assume it is at the destination.

Let's combine all the pieces



Job 9-1: Provide field definitions in DREAMM

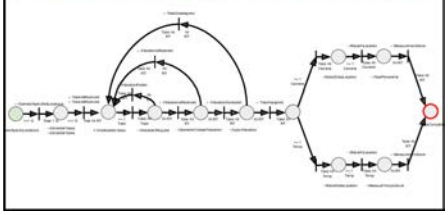


- The SPN is already provided
- Provide definitions for all output and input event's fields
- When defining variable names, use whatever name you would like

The final step in completing the SPN is setting an end place.

1- Get the locations
2- Create the task tokens
3- Get task allocation options
4- Choose an allocation
5- Apply the chosen allocation
6- Perform and complete the assigned tasks
7- End the plan when all tasks have been completed

QUIZ 9-4: What would happen if this was the end place?



Quiz 9-4 Solution

The plan would end as soon as the first task was completed

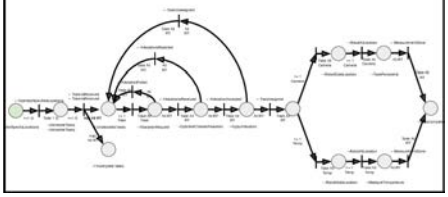
As a reminder, we tell the system when a single task is completed by activating a →TaskComplete output event with that task token:
→TaskComplete: Tells the system that a Robot has finished its assigned task
- No fields

However, we don't have a way of knowing when all tasks have been completed.

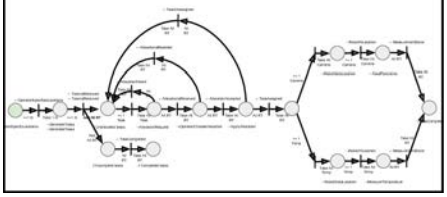
One way of doing this is to have a place that receives one copy of each task token as they are created. When that task has been completed, it consumes or takes that task token out of the place. When the place has no task tokens left, we know all tasks have been completed.

To do this, we will need a new input event:
←TaskCompleted: Indicates a task was marked as being complete via →TaskComplete
- No fields
- Relevant Tokens: The token for the task which was completed

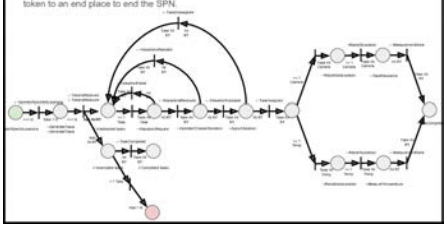
Here we make a new place labelled "**// Incomplete tasks,**" which also gets each of the ←**TokensReceived** input events' relevant tokens added to it, just like the "**// Unallocated tasks**" place.



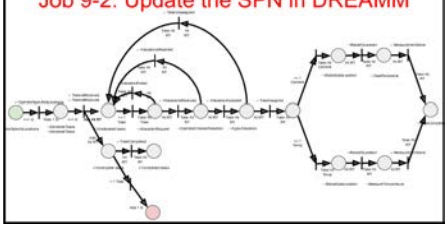
When we receive a ←**TaskCompleted** input event, we move the relevant token (the completed task's task token) into the "**// Completed tasks**" place. Alternatively, we could consume the token.



When the "**/ Incomplete tasks**" place has no more task tokens, we are done and can add a Generic token to an end place to end the SPN.



Job 9-2: Update the SPN in DREAMM



Markup

Markup is additional information attached to input and output events in a plan. This contextual information can modify how the event is processed by the system, usually in the following ways:

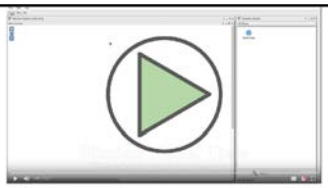
- Changing what or how information is shown in the GUI
- Changing how involved the operator is in decisions

The first markup we will look at is RelevantArea. It is used to ensure a certain area of the map is visible to the operator when a particular input event is received or output event is activated.

RelevantArea has one piece of information to fill out:

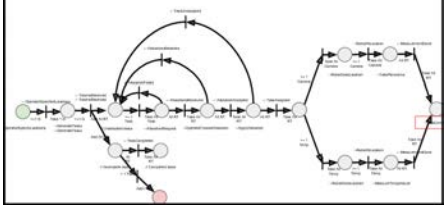
Area: What locations should be be visible in the map? This may cause the map to be zoomed out and panned to make the locations visible. There are 3 options:

- Area: A specific area should be visible on the map.
  - If you select Area, you will need to specify the Area
- All Robots: All robots should be visible on the map.
- Relevant Tokens:
  - If placed on an input event: For all Relevant Tokens which are robot tokens, make sure the the robot is visible. For all Relevant Tokens which are task tokens, make sure the allocated robot (if one exists) is visible.
  - If placed on an output event: For any robot token which activates the output event, make sure the robot is visible. For any task token which activates the output event, make sure the allocated robot (if one exists) is visible.

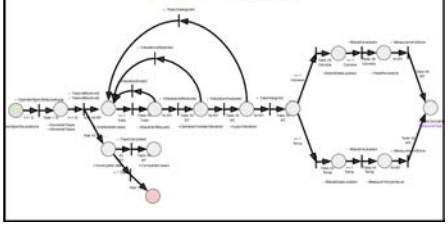


Watch "Adding Markup": This video will show you how to add markup to an existing event.

Job 10-1: Add RelevantArea markup to →TaskComplete using the Relevant Tokens option



Job 10-1 Solution



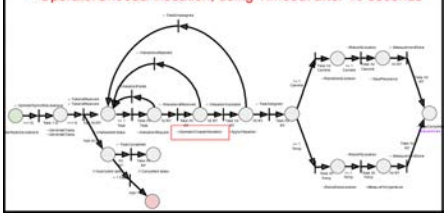
The second markup we will look at is Mixed Initiative. It is used on output events which involve the operator making a decision. Using this markup can give AI permission to make decisions when the operator is too busy, allowing the plan to progress. Some decisions may be trivial and can always be always given to the AI. Others may be very complex and should only be given to the operator, even if it means waiting until the operator isn't busy.

MixedInitiative has one piece of information to fill out:

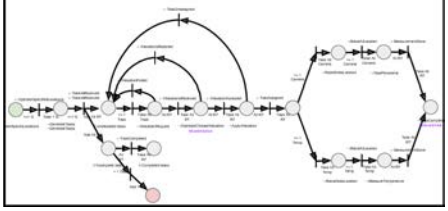
Trigger: When should the system AI make the decision instead of the operator? There are 3 options:

- Never: Never allow the AI to make this decision.
- Immediate: Do not show the decision to the operator, let the AI decide immediately.
- Timeout: Show the decision to the operator, but remove it after x seconds and let the AI decide.
  - If you select Timeout, you must specify the number of seconds.

Job 10-2: Add MixedInitiative markup to →OperatorChooseAllocation, using Timeout after 10 seconds



Job 10-2 Solution



Final Plan

Job 11-1: Now you will need to construct a new plan which does the following

- Feel free to reference your existing plans and previous lessons as it uses elements from them.
- Don't forget to provide values or variable names for event fields

First, the operator should be asked to create a list of locations

When the list of locations is received, Camera tasks should be generated from the list of locations

When the task tokens are received, they should be allocated immediately by the System AI.

When a task is assigned, the robot for the task should move directly to the task's location

When the robot for the task arrives, it should take a panorama

After taking the picture, the robot should wait 5 seconds

After waiting 5 seconds, the task is complete

When there are no unfinished tasks, the mission should end