Mission Modeling, Planning, and Execution Module for Teams of Unmanned Vehicles

Jean-Pierre de la Croix†, Grace Lim†, Joshua Vander Hook†, Amir Rahmani†,
Greg Droge‡, Alexander Xydes‡, and Chris Scrapper‡

†California Institute of Technology, Jet Propulsion Laboratory, Pasadena, CA
‡Space and Naval Warfare Systems Center Pacific, San Diego, CA 92152

ABSTRACT

Mission Modeling, Planning, and Execution Module (M2PEM) is a user friendly graphical framework for mission design and execution. It extends a subset of the Business Process Modeling and Notation (BPMN) 2.0 for robotic applications. Hierarchical abstractions fundamental to BPMN allow the mission to be naturally decomposed into interdependent parallel sequences of BPMN elements. M2PEM adapts these elements in a role based framework which uses collaborative control modalities as an atomic building block. Designed missions are able to consider situational data, external stimuli, and direct user interaction. Missions are directly executable using a resource manager and a ROS-based execution engine.

Keywords: Robotic Mission Planning, Mission Executive, BPMN

1. INTRODUCTION

As unmanned systems proliferate more widely into the Navy, a need arises for a simple and user-friendly method of controlling a large number of these systems with a small number of (perferably a single) operators. These operators need to control large groups of unmanned systems while ensuring that operationally relevant behaviors are executed. At the same time, one must verify that mission objectives will be met, ensure mission longevity via ensuring adaptability and modularity, and enforce well-defined interactions with unmanned systems. A tactical mission control approach needs hierarchical abstractions to provide modularity and limit information overload. It needs parallel sequencing to allow for parallel exeuction of tasks and objectives. And finally it needs defined and deliberate user interaction to allow operators to influence the mission during execution.

Within collaborative control methodologies, swarm or single behavior control is either highly reactive and/or focused on a single action (e.g. formation control, area coverage, positioning, etc). One example of formation control and positioning uses potential fields to control the spacing of unmanned systems.[1] Another approach uses task-based single-vehicle allocation where each vehicle is assigned to one task at a time.[2] While these swarm control methodologies lead to emergent behavior, they lack a more deliberative approach that that provides for longer-term planning and decision making. A tactical mission control approach should be able to leverage these swarm behaviors while also providing more deliberative planning and control of the mission.

Some of the more deliberative approaches coordinate simple sequences of behaviors for a single unmanned system,[3–5] or use a Finite-State Machine (FSM) to represent the mission of the team of unmanned systems.[6] That particlar approach (by Ron Arkin et al.) includes a visual programming tool, cfgedit, that can define robot missions using a Finite State Machine (FSM) representation with calls to underlying behaviors.[6–9] They also provided verification tools for such missions based on Linear-Temporal-Logic (LTL) representation of the desired outcome. FSM-based missions are validated against these LTL specifications to ensure robots are "getting it right the first time."[10,11] However, FSMs do not natively support parallel states of execution which are necessary for tatically and operationally relevant behaviors.

Further author information:
†: {Jean-Pierre.de.la.Croix, Grace.Lim, Joshua.Vander.Hook, Amir.Rahmani}@jpl.nasa.gov
‡: {alexander.xydes, chris.scrapper}@navy.mil
‡: {greg.droge}@us.af.mil

Unmanned Systems Technology XIX, edited by Robert E. Karlsen, Douglas W. Gage, Charles M. Shoemaker, Hoa G. Nguyen, Proceedings of SPIE Vol. 10195, 101950J · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2266881

Another popular approach to mission design is to define a custom language for specifying different parts of a mission, desired behaviors to be called and their parameters. An example is the work by Perdomo et al., that uses an XML based custom language to specify missions for underwater robots.[12] As versatile as this approach can be, it is not very user friendly for large missions and also does not lend itself to other mission types on different (e.g., aerial) platforms.

Mission Modeling, Planning, and Execution Module (M2PEM) leverages business process logic to coordinate tactical behaviors and mission objectives between a heterogeneous team of unmanned systems. In order the develop this logic M2PEM leveraged Business Process Modeling and Notation (BPMN) 2.0[13] to develop mission models by extending a subset of BPMN 2.0 for robotic applications. Hierarchical abstractions fundamental to BPMN allow the mission, designed graphically, to be naturally decomposed into interdependent parallel sequences of Activities, Gateways, and Events. Figure 1 illustrates an example mission encoded as an M2PEM mission model.

The graphically defined mission is encoded as a mission model diagram and stored in a machine readable (BPMN XML standard) format. It is parsed and processed by the M2PEM executive. The executive consists of a resource manager used to assign resources to roles within each activity and an execution engine for commanding the behavior defined by each activity to the appropriate robots and/or resources. This paper describes the architecture of M2PEM, the interface used to build a mission model, mission execution, mission verification and field testing of the system.

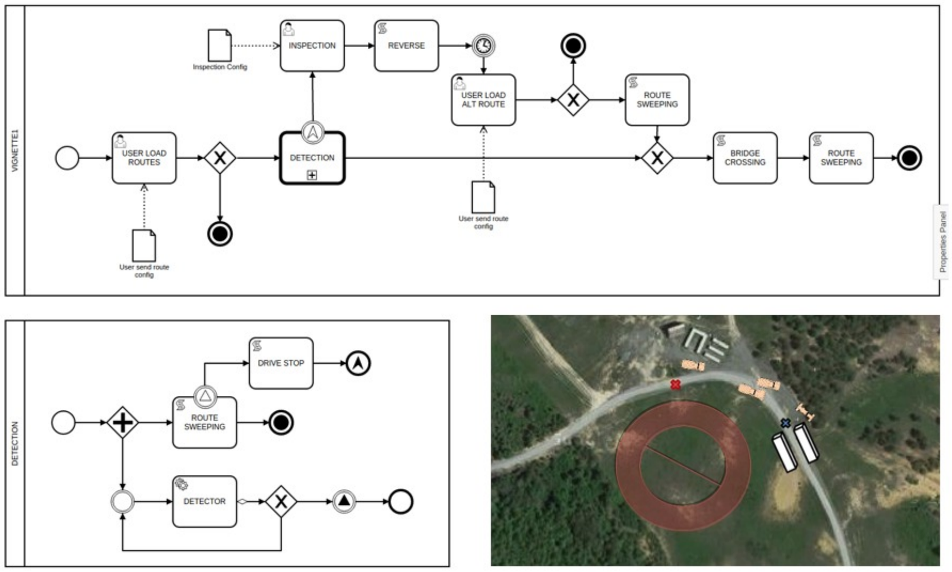


Figure 1: An example mission encoded as an M2PEM mission model.

2. M2PEM ARCHITECTURE

In most robotics applications or robotics software development environments, missions have to be implemented or described in code or via configuration files.[12] This approach is not only complex and cumbersome for a developer of this system, but also is inaccessible to someone without direct domain knowledge. Graphical tools, such as BPMN, are easy to understand constructs for describing a sequence of actions, or more specifically, the flow of a process. For example, an organization could use BPMN to describe how a customer service representative should handle a product return. Similarily, we would like to model how a team of multiple robots should handle a detection from a sensor during a mission.

Our solution is to use BPMN to graphically describe missions for multi-agent robotic systems, while providing an execution and verification engine for autonomously running these missions on the system. M2PEM uses the BPMN 2.0 specification to allow a user to plan out the mission for a system of multiple robots. Extensions to the BPMN 2.0 specification in M2PEM correspond to special constructs that are applicable to a robotic system and are explained further in Section 3. Currently missions are scripted, with static resource allocations, which means that vehicles and payloads are assigned to roles for each activity within the mission at design time. These allocations can not be changed during execution, and the mission will fail if a vehicle or payload can not be found by the resource manager.

M2PEM features an execution engine to directly control the vehicle and payloads in the system (via interfaces to the Robotics Operating System (ROS)[14]). M2PEM also features a verification module, which is capable of checking for erroneous logic in the mission (such as dead/live locks), as well as verify mission specifications (such as, "each robot has to survey at least two locations"), which can be expressed in Linear Temporal Logic (LTL).

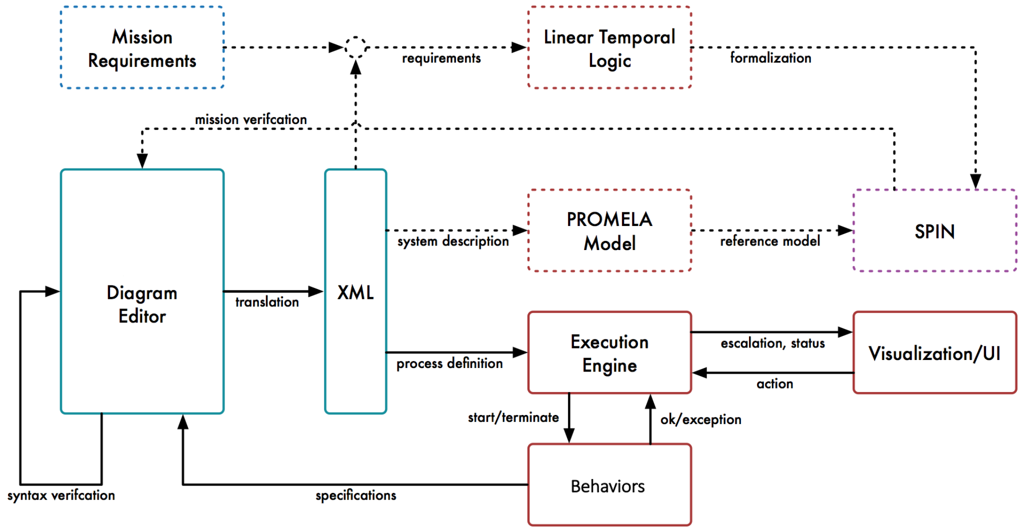


Figure 2: M2PEM architecture: Solid items represent *execution* elements, while dashed items represent *verification* elements.

Figure 2 depicts the architecture of the M2PEM software. A *graphical user interface* (Camunda[15]) is used to create and edit a *mission model*. This mission model is stored as a standard BPMN XML *mission file* that is passed to the *execution engine*. The execution engine parses this mission file and, if it passes the syntax and parsing checks, it prepares for execution. Once signalled to start executing, the execution engine then systematically moves through the mission model, following the flows and calling vehicle and payload behaviors corresponding to active mission blocks. The user interface doubles as a visualization of mission progress, highlighting currently active blocks.

To verify the mission model, the mission model file is translated to the Process Meta Language (PROMELA)[16] and checked against the mission requirements, encoded as LTL specifications, using the Spin model checker.[17] We discuss our progress on this verification module in more detail in Section 5.

3. MISSION MODEL AND USER INTERFACE

A mission is the highest level description of what robots should accomplish when they are deployed. In M2PEM, missions are encoded as *mission models*, which are a set of processes required to complete a particular mission. M2PEM currently uses a subset of the BPMN 2.0 standard to define the activities and process flow (collection of Tasks and their transitions) inherent in a mission model. For ease of use, a graphical model of the mission is created by the operator using notions defined by BPMN. This *mission model diagram* has a one-to-one equivalent BPMN standard XML representation, dubbed the *mission model file*.

As mentioned, a relevant subset of BPMN was chosen for initial implementation, however new elements are added as needs arise. This subset includes process, call activity, user task, script task, service task, gateway, and multiple types of events. Elements included in the events category include signals, escalations, and boundary events. Each process contains an interdependent, potentially parallel, sequence of activities, gateways and events. In M2PEM, activities are composed of atomic behaviors consisting of collaborative control modalities using a role-based framework for defining required and optional resources for execution.

Activities include subprocesses, call activities, and tasks. Subprocesses are equivalent to a process, except they ecapsulate the entire sequence within one activity and can be maniuplated like the other activities. They can also only be defined within other processes, and can access all of the parent process's data. In addition, they can only have plain start events, no other type like timer or message based start events. Call activities are very similar to subprocesses, except they are defined globally (i.e. not within another process) and referenced within other processes. They also do not implicitly have access to the parent process's data, nor are they limited to plain start events.

Tasks represent individual activities to be executed, and can be of many types. M2PEM primarily uses script, service and user tasks. Script tasks contain a piece of code that is to be executed or interpreted directly by the execution engine. Service tasks are linked to specific C++ class objects that are executed. User Tasks allow a user to directly interact with the autonomous system while it is executing its mission. User tasks are configured via a data file object in the mission model. This data file object includes a *prompt* to provide to the user and a series of *decisions* to choose from.

Data-driven Gateways are used to access situational data, as well as change the flow of the mission. They can also split the mission into parallel execution flows or merge multiple parallel flows back into one flow. Events allow for external stimuli to influence process flow, and throw/catch type events all contain a unique identifier or UUID. When execution flow hits a throw event, the unique identifier is used to look up any catch events of the same type and if one or more are found the execution flows to those events.

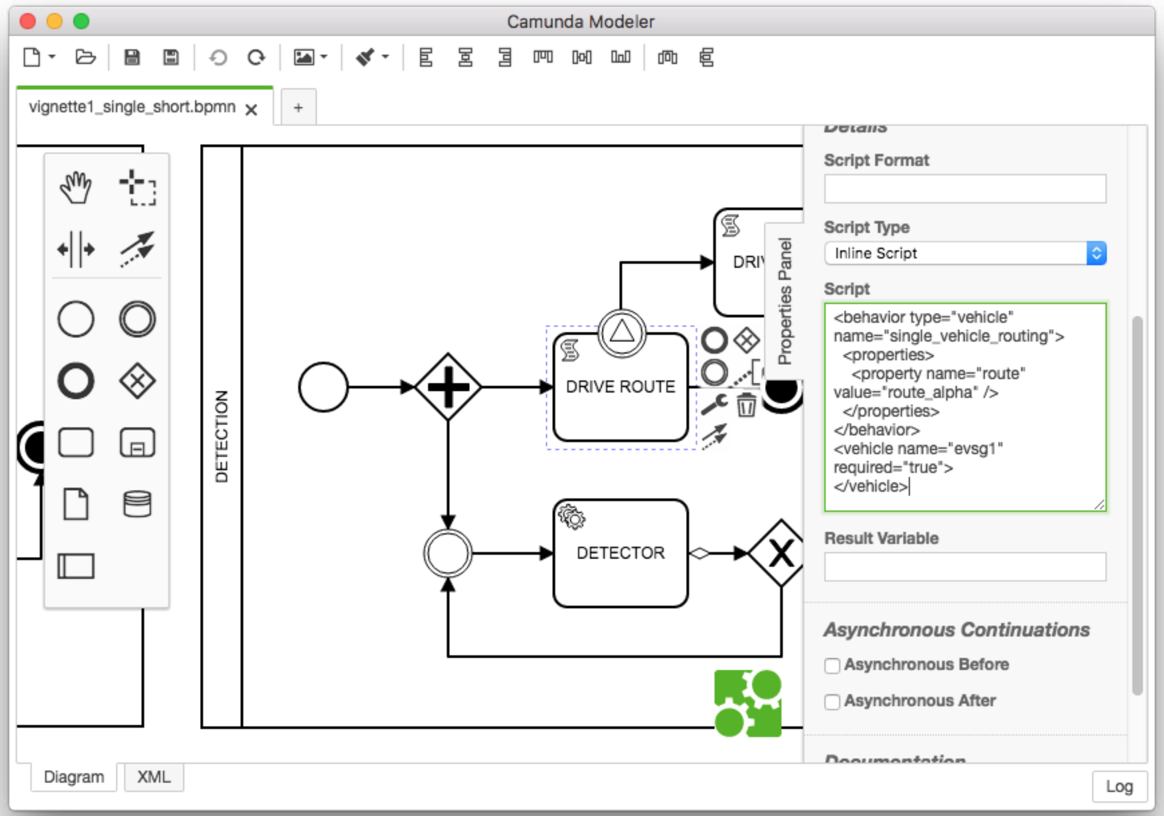
We also adopted some of the standard BPMN elements and defined them to represent robotic activities of interest to our mission domain. Data-driven gateways can access contextual data, such as user decisions (e.g., use the route named `route_alpha`), to control the flow through the mission. Tactical behaviors are specified by using a *script task* with an XML script, because BPMN 2.0 is general enough to allow for special cases. That XML script defines whether it's a vehicle or payload task as well as which behavior to call, which resources to use and any parameters that the behavior needs. Vehicle tasks are those that are assigned to one or more vehicles, which call an individual (e.g., *waypoint navigation*) or team behavior (e.g., *formation control*) on the vehicles. Payload tasks, on the other hand, will call payload behaviors (e.g., PTZ *camera scan polygon*) on different payloads available on vehicles. Figure 3 illustrates how the script task labeled `DRIVE ROUTE` invokes a behavior for the vehicle to follow a particular route.

The XML tag `behavior` tells the execution engine that this script task is going to either be of Vehicle or Payload task type. The XML property `type` defines the type of behavior task and is used by the behavior engine to address the proper arbiter via a specific ROS topic. The `name` property corresponds to one of the behaviors supported by the vehicle arbiter. The `properties` tag specifies all of the parameters for this behavior as a list of `property` tags. In this particular example, the `single_vehicle_routing` behavior is informed to use a route named `route_alpha`. The other information provided to this script task is the static resource allocation, i.e. which vehicle and payload to use for each role, which in Figure 3 is a single vehicle named `evsg1` (a simulated vehicle).

Camunda,[15] an open source `node.js` based BPMN editor, was selected as the standard user interface for defining a mission. We have customized this editor to allow for custom activity blocks, and provide easy to edit parameter selection for those custom blocks. Other improvments made include highlighting parsing and verification errors, and visualizing mission progression using feedback from the execution engine over a custom WebSocket interface. Figure 4 shows how our custom visualization highlights the elements currently active. Current elements are bright green when active but become gray and fade as a function of time after deactivation. This gradient based approach is chosen to give user a chance to follow the execution path for fast moving mission flows.

4. MISSION EXECUTION

Mission execution is the component of the M2PEM architecture that interprets the mission plan (XML-formatted BPMN) and follows its sequence of flows from element to element. Conceptually, the BPMN 2.0 standard defines



Details

Script Format

Script Type: Inline Script

Script

```
<behavior type="vehicle"
name="single_vehicle_routing">
  <properties>
    <property name="route"
value="route_alpha" />
  </properties>
</behavior>
<vehicle name="evsg1"
required="true">
</vehicle>
```

Result Variable

Asynchronous Continuations

☐ Asynchronous Before

☐ Asynchronous After

Documentation

Figure 3: Script task details expanded in the mission model.

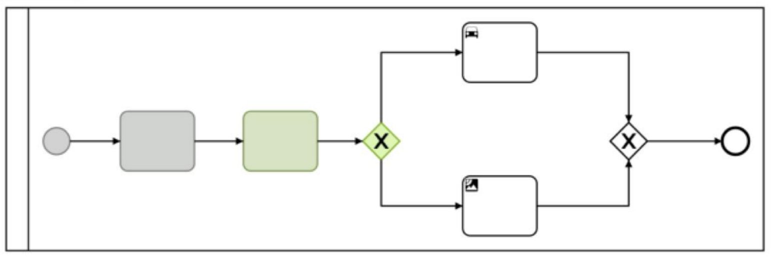


Figure 4: Custom visualization of current active element(s).

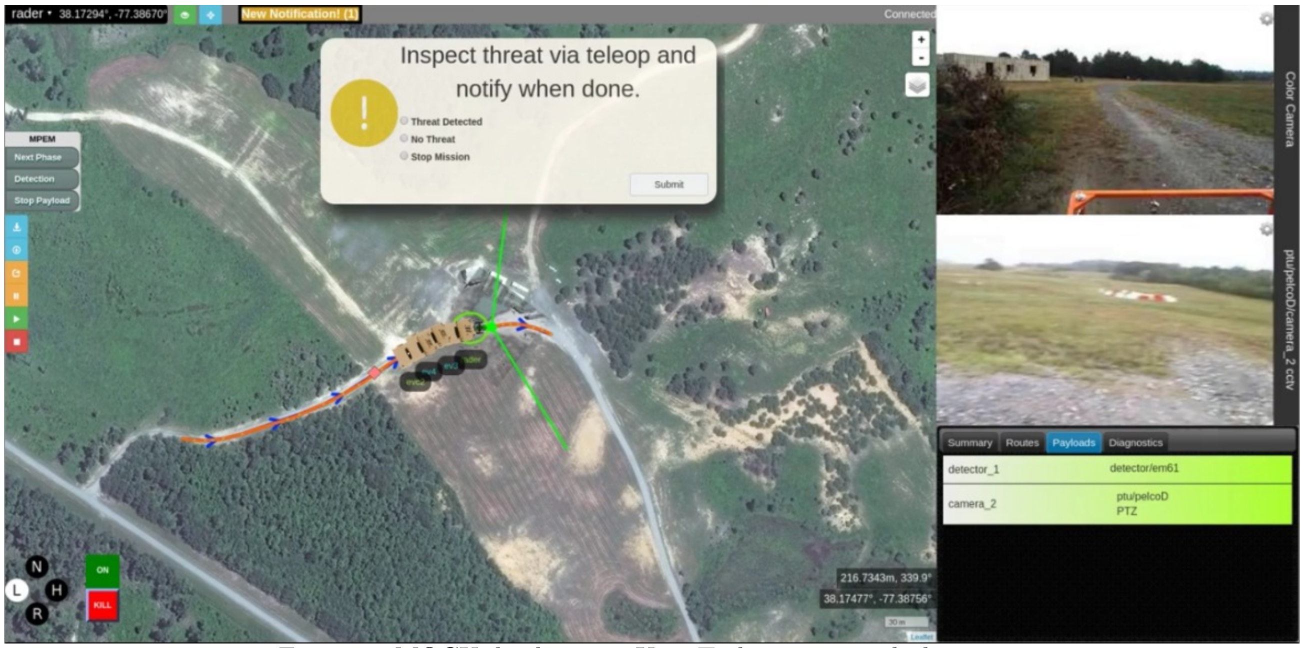
tokens, which are created at start events or whenever flows diverge (for example, at parallel gateways) and are either consumed at end events or cancelled. The execution engine implements tokens (although this is not a requirement of the BPMN 2.0 standard) to provide the parallel threads of execution required to handle executing several mission model elements simultaneously. This section focuses on how the execution engine uses tokens to execute a mission model on a robotic platform.

4.1 Execution with Tokens

Before the mission is executed, a parser interprets the BPMN 2.0 mission model (an XML file) and creates an internal representation of this model. Each element in the model is of a base type `ProcessModelNode` and is linked to other elements via `SequenceFlow` objects. Effectively, the mission model is internally represented as a hierarchical graph data structure. Upon being signaled to start (via a ROS service call, as described in Section 4.3), the execution engine generates a token to own the first process, whose id is passed in via ROS parameter. This token executes the main function of a `Process`, which is to spawn child tokens for each start event (in this case, there is only one). For plain start events, the token immediately moves to the next element via the attached `SequenceFlow` objects. If the start event is not a plain start event, the token assigned to it would wait for that event's conditions to be fulfilled before moving via the `SequenceFlow` objects to the next node.

Among the different Activities, subprocesses are treated very similarly to a process, with the token that is assigned to the subprocess spawning a new token for the start event contained within. Once the token started within the subprocess reaches the end event, it is consumed and the token assigned to the subprocess can proceed to the next node. Call activities are treated similarly to subprocesses, except that the token assigned to it calls the global process whose ID is specified by the call activity. Of the task activities, script tasks are treated special by M2PEM during execution as explained further in Section 4.2. When a token reaches a service task, the execution engine creates an instance of the class specified by the task. Then the engine executes the instance it created and waits for it to finish.

For user tasks the execution engine advertises the prompt and decisions on a ROS topic that is being listened to by the Operator Control Unit (OCU). The Multi-Robot Operator Control unit (MOCU) has been modified to interact with M2PEM, allowing for interaction via user tasks. MOCU displays the prompt and decision choices to the operator, who selects one of the decisions. Upon having submitted a decision, MOCU uses a ROS service call to inform the execution engine of the user's decision. This decision is stored in the execution engine's internal data store (a SQLite database). This action concludes the user task and the child token proceeds to the data-driven exclusive gateway. Figure 5 is one example of how a user decision is displayed to the operator during a mission.


Inspect threat via teleop and notify when done.

Threat Detected
No Threat
Stop Mission

Figure 5: MOCU displaying a User Task prompt with decisions.

Data-driven gateways will send the current token along a sequence flow that evaluates to true and spawn additional tokens for each additional outgoing sequence flow that evaluate to true. Sequence flows without a condition are always true, while sequence flows with conditions are evaluated to check if at this instance they are true or false. These conditions typically correspond to checking whether a particular value is equal to some constant. Exclusive gateways only select a single sequence flow, but if multiple flows evaluate to true then it selects the first one specified in the XML.

When a token reaches a timer event, it waits for the specified amount of time before moving on to the next node. Throw and catch events cause tokens to spawn on nodes not necessarily connected by a `SequenceFlow` objects, and can also cause tokens waiting for a node to interrupt it and move on to a new node. This mechanism uses ROS topics to message between the tokens.

4.2 Script Task

Since some of the tasks in a process model are automated, commercial BPMN engines tie those tasks via a connector to a web service that can handle a request and respond with a result. Similarly, the execution engine in M2PEM requires a method by which it can invoke automated tasks on the hardware platform. Specifically, we want to be able to invoke vehicle and payload behaviors. Arbiters in the unmanned system are responsible for providing a service-like API to invoke behaviors. They are commandable via a ROS topic with a specific ROS message type and provide status updates on a separate ROS topic.

The execution engine first uses the static allocation specification within the script task to get an allocation from the resource manager. The resource manager uses a resource graph that contains availability information on all vehicles and payloads to validate the static allocation. If the vehicle is available, it then addresses the arbiter via a predefined ROS topic and populates a command message with the behavior name and key-value pairs for all of the parameters. The script task then monitors a specific ROS status topic advertised by the arbiter to check if the behavior is running. As long as the arbiter is running the behavior, the script task remains active. If the script task detects that the behavior is no longer running (could be due to the behavior completing, or the behavior was aborted), then it either completes or throws an exception depending on the outcome of the behavior. Exceptions can be caught by boundary error events and handled in the model, otherwise they are propagated to the parent process until they are handled or abort the mission.

4.3 Mission Control

External to the mission execution, the operator using MOCU has the ability to interact with the mission. Specifically, MOCU has the ability to start and stop a mission. Starting a mission requires a file location (URL) to the mission or use of the M2PEM editor to directly send the BPMN file. Stopping a mission will cascade through the mission plan and terminate all tokens. Arbiters with active behaviors are notified of this termination.

MOCU is also able to provide the execution engine with waypoints or a route that can be referred to by the mission model. For example, a series of waypoints can be stored as `route_alpha` and used as a parameter for a vehicle behavior. Signals (with a specific UUID) can also be emitted from MOCU and are caught in the executive engine. These signals may, for example, be used to abort specific portions of the mission without stopping the entire mission.

All mission control functionality is achieved using ROS services. This provides a standard interface to the execution engine and allows for multiple OCU's to interact with it in the same manner.

4.4 Execution Example

This next section walks through the execution of a simple example mission by M2PEM. For this purpose, we will refer to the mission model illustrated in Figure 6, which includes many of the elements that were exercised during field testing. These include custom user tasks that interact with MOCU, script tasks that command the vehicle or payload arbiters, and service tasks that invoke custom code blocks. As well as other BPMN 2.0 elements such as timer events, escalations, and data-driven gateways.

Upon being signaled to start the execution engine generates a token to own the process labeled `VIGNETTE1` in Figure 6. That token then generates tokens for each start event within the process, of which there is only one. Since this start event does not have an event definition, the child token immediately uses the sequence flow to move onto the user task labeled `USER SEND ROUTE`.

At this point, the execution engine advertises the prompt and decision choices to MOCU and waits for the operator's response. In this case, the operator must also make sure that the M2PEM execution engine has access to the appropriate route required by the `DRIVE ROUTE` script task later in the mission. Once the operator's response is recorded into the internal data store, flow moves on to the exclusive gateway. For this gateway, each

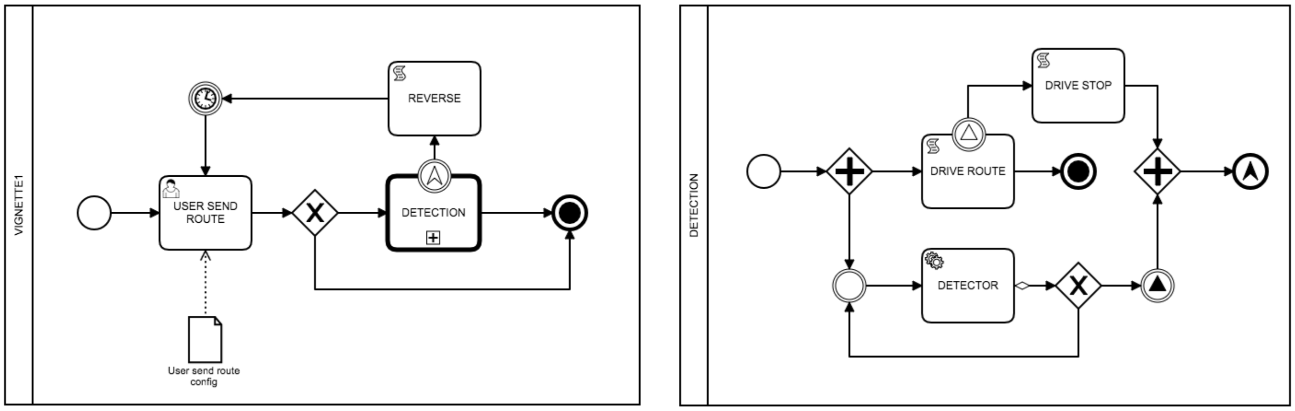


Figure 6: An example mission encoded as an M2PEM mission model.

outgoing flow checks whether the user decision (from the previous user task) is equal to `continue` or `abort`. The evaluation is achieved by looking up the value of `user.decision` stored in the data store.

If `user.decision` equals `abort`, the token would follow the sequence flow that bypasses the next block and arrive at the terminating end event (which would conclude the mission). Suppose that `user.decision` instead equals `continue` and the token moves onto the call activity labeled `DETECTION`. This call activity spawns a new token for the process labeled `DETECTION`. This subprocess spawns a new token for its start event, which then arrives at a parallel gateway. A new token is generated to follow the path that leads to the service task labeled `DETECTOR`, while the token processing the parallel gateway moves onto the script task labeled `DRIVE ROUTE`.

When the `DRIVE ROUTE` script task concludes, the token will arrive at a terminating end event for this subprocess, which would then allow the token in the main process on the call activity to proceed to the terminating end event of the main process and conclude the mission. However, this particular script task also has an attached boundary signal event. If the signal event is triggered, the token proceeds along the alternate path to another script task labeled `DRIVE STOP`.

Switching to the bottom half of the subprocess, the other token spawned from the parallel gateway is executing the service task labeled `DETECTOR`. In this case, the `Detector` class is called, which listens to a ROS topic for a detection message and writes into the data store information about the detection including the detection source device (for example, `detection.type` is `gpr` for ground penetrating RADAR). The following data-driven exclusive gateway has two outgoing sequence flows with a condition to check if `detection.type` is equal to `gpr` or not. If not, the token proceeds along a path that returns it to the service task. Otherwise, the token proceeds to an intermediate signal throw event.

In this mission model, the signal UUID of the intermediate signal throw event is equal to the boundary signal event on the script task labeled `DRIVE ROUTE`. Consequently, if there is a token on the script task and another token arrives at this intermediate signal throw event, then the boundary event on the script task will be caught and processed. If this situation occurs, the token on the intermediate signal throw event proceeds to the parallel gateway and is consumed, while the token on the script task follows the path along the boundary event to also arrive at the same parallel gateway. Since all pathways to the parallel gateway have been activated, this token proceeds to the end event, which emits an escalation.

This escalation event is caught by the boundary event on the call activity. Consequently, the token on the call activity follows the path along the boundary event to the script task labeled `REVERSE` and eventually back to the original user task labeled `USER SEND ROUTE`. The mission continues until the terminating end event is reached in the main processes labeled `VIGNETTE1`.

5. MISSION VERIFICATION

The goal of mission verification is to check the mission model for any errors and against any *mission requirements*. Our approach to verification uses the Spin model checker,[17] a tool for analyzing the logical consistency of concurrent systems described in the Process Meta Language (PROMELA).[16] Spin was chosen over other model checkers due to being freely and actively maintained since 1991, its history of use in flight projects, use of Linear Temporal Logic (LTL),[18] and ability to generate C++ binaries for problem specific model checkers that are fast and memory efficient. Since Spin operates on PROMELA models, we first automatically translate BPMN mission models into Promela models using a combination of an XML parser and predefined templates for each BPMN element. Currently, we support a small number of BPMN elements–tasks, signals, (sub)processes, call activities, and parallel gateways. A set of LTL formulations are also included to check for safety (e.g., deadlocks and mutual exclusion) and liveness properties (e.g., starvation and deadlocks). Spin is run with the generated PROMELA model files and any LTL specifications, and we manually check the output to verify that the mission is valid. Currently, we have tested simple LTL specifications (i.e., mission requirements), such as, "Does the mission reach a proper end state?", which in LTL can be expressed as, "Is the end event eventually true (active)?"

6. FIELD TESTS

M2PEM was successfully demonstrated in multiple test events at Camp Pendleton and Fort A.P. Hill, commanding a single as well as a team of autonomous High Mobility Multipurpose Wheeled Vehicles (HMMWVs). Vehicle and payload arbiters provided a host of section-level and vehicle-level behaviors; Road-following Formation, Road Sweep, Bridge Crossing, and Area Observe are examples of behaviors commanded by M2PEM during the demonstration events. Figure 1 depicts a sample mission model executed by M2PEM. Figure 7 depicts two vehicles executing *formation control* task. The M2PEM framework allows for defining a scan task for the PTZ camera on the aft vehicle while it is in formation with the lead vehicle.

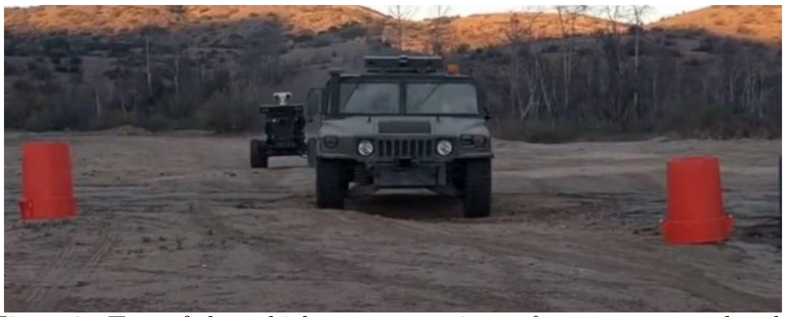
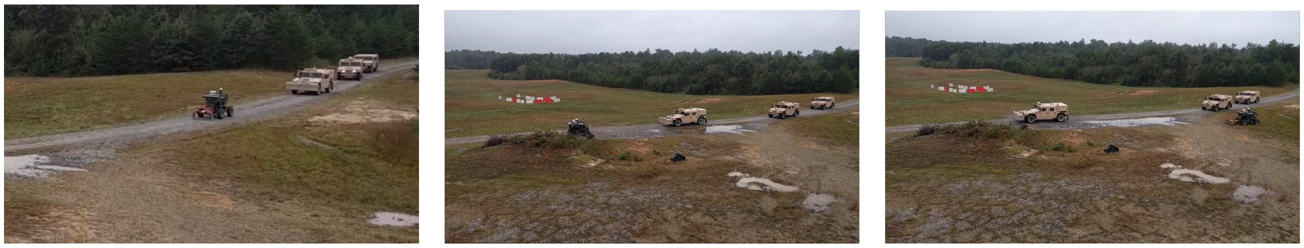


Figure 7: Two of the vehicles are executing a *formation control* task.

M2PEM provided the section-level planner for the Modular Explosive Hazard Defeat System (MEHDS) demo that took place at Fort A.P. Hill in September 2016. There were four vignettes designed to showcase multiple situations involving detection and defeat of explosive hazards. The first involved switching routes after discovering and inspecting an explosive hazard blocking the first route, then crossing a bridge. Vignette 2 showcased the maneuverability of the platforms, and had them split up then join back together. The third vignette involved inspecting a bridge that was deemed suspicious, discovering an explosive hazard under the bridge, then choosing a new route and having a mine roller in front clear the route. The last vignette starts with normal route following, then after a triggerman is discovered, has a platform with a ground-penetrating radar (GPR) take the lead and stop the formation once an explosive hazard is discovered.

The first vignette included four vehicles, three autonomous and one manned, and three main parts to the mission (Appendix A). The mission starts with the main route sweeping and detection section (Figures 8a, 12). After a detection occurs (Figure 8b), M2PEM throws a signal which causes the token at "route sweeping" to move to "drive stop" and then escalates out of the main detection subprocess (Figure 12). The token then moves into the inspection area of the mission (Figure 9) where the operator teleops a second vehicle over to the detection site to get a second confirmation (Figure 8c). After the inspection and alternate route traversal, the mission calls for operator confirmation before crossing the bridge (Figure 10) for which M2PEM sends a signal to MOCU and waits for the response. Once the operator has confirmed that they want to cross the bridge, M2PEM enters the bridge crossing behavior and then ends the mission with a route sweeping afterwards.


(a) Route sweeping (b) Explosive hazard detection (c) Explosive hazard inspection
Figure 8: Vignette 1 highlights

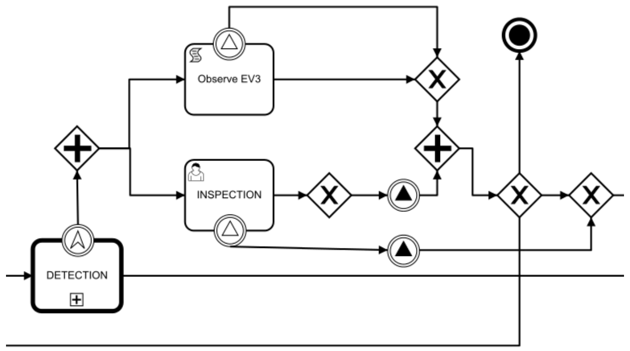


Figure 9: Closer view of the inspection part of the vignette 1 mission.

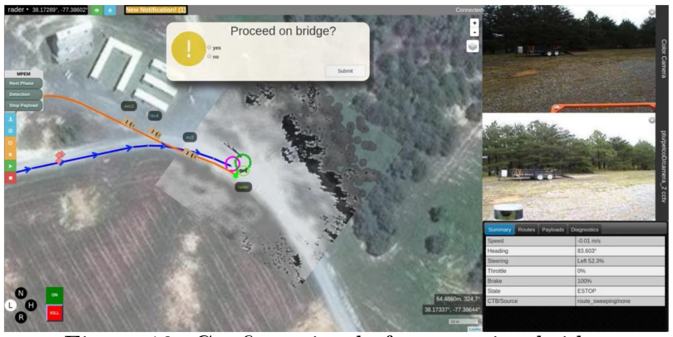
7. FUTURE WORK

One of the goals for future work is adding dynamic resource allocation to the resource manager. This will allow M2PEM to adaptively assign assets to roles within the mission at execution time. As well as help it cope with resource failure by allowing it to change which asset is assigned to which role when an asset fails.

In the future, we plan to provide the verification output from Spin in our custom BPMN modeler (or in the OCU) to highlight any issues in the mission model for the operator. We also plan to expand the set of supported BPMN elements, specifically data-driven inclusive and exclusive gateways, event-based gateways, and data artifacts (data stores and objects). We will also test a richer set of LTL mission requirements to demonstrate the efficacy of the verification module.

8. CONCLUSIONS

M2PEM is a powerful, user friendly and multi-mission capable mission executive. Compared to BPMN (or other modeling languages), M2PEM provides not only extensions that are specific to robotics, but more importantly, a ROS-compatible execution engine that can interact with actual hardware. Compared to other mission planners,



Proceed on bridge?

yes
no

Speed	-0.01 m/s
Heading	83.603°
Steering	Left 52.3%
Throttle	0%
Brake	100%
State	ESTOP
CTB/Source	route_sweeping/none

Figure 10: Confirmation before crossing bridge

M2PEM's mission can be described graphically without writing code or configuration files. M2PEM effectively brings BPMN into a robotics application, which is novel. Other state of the art approaches include writing LTL or drawing state diagrams of the missions. While these are feasible and demonstrated approaches, using BPMN for modeling and notation is somewhat more powerful than state machines, because BPMN supports concurrency, information flow, signals, exceptions, and other constructs that are not describable in standard state machines. Consequently, the novelty of M2PEM is that it is capable of supporting more complex missions (including those for multiple vehicles and payloads). M2PEM can easily be configured for other vehicles types and application domains; these are topics of our on-going work along with standardization of the executive and arbiter interfaces for heterogeneous and mutli-domain missions.

ACKNOWLEDGMENTS

This work was performed as a collaborative effort between the Jet Propulsion Laboratory, California Institute of Technology, and SPAWAR System Center, Pacific under sponsorship by the Office of Naval Research (ONR) Code 30. The views expressed are those of the authors and do not reflect the official policy or position of the funding agency or the U.S. Government. Copyright 2017. All rights reserved.

APPENDIX A. VIGNETTE 1 MISSION PLAN

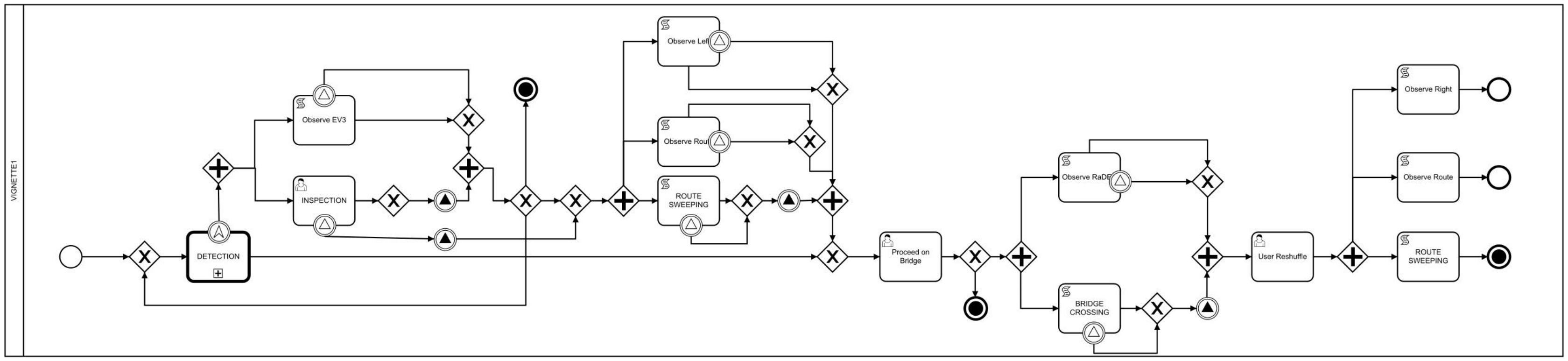


Figure 11: Vignette 1

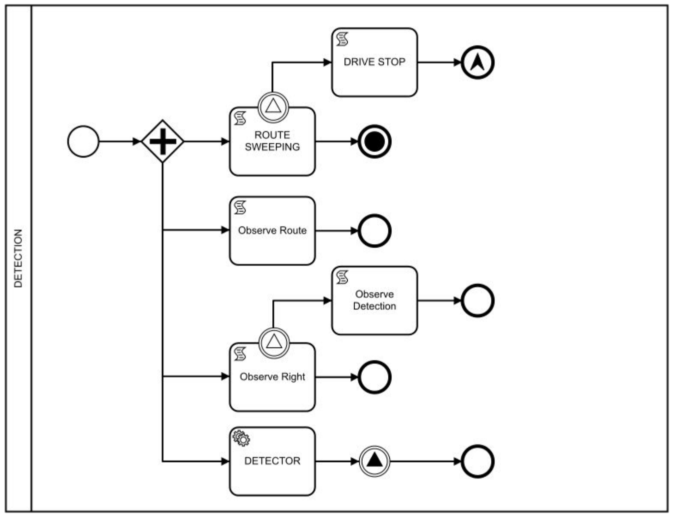


Figure 12: Vignette 1 detection process

REFERENCES

[1] Barnes, L., Fields, M., and Valavanis, K., "Unmanned ground vehicle swarm formation control using potential fields," in [*2007 Mediterranean Conference on Control Automation*], 1–8 (June 2007).

[2] Schneider, E., Balas, O., Ozgelen, A. T., Sklar, E. I., and Parsons, S., "An empirical evaluation of auction-based task allocation in multi-robot teams," in [*Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*], *AAMAS '14*, 1443–1444, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2014).

[3] Crockett, T., Shams, K., and Morris, J. R., "Telerobotics as programming," in [*2011 IEEE Aerospace Conference*], 1–7 (March 2011).

[4] Powell, M. W., Shams, K. S., Wallick, M. N., Norris, J. S., Joswig, J. C., Crockett, T. M., Fox, J. M., and Torres, R. J., "MSLICE Science Activity Planner for the Mars Science Laboratory Mission," *NASA Tech Briefs* , 51–52 (Sep 2009).

[5] Sherwood, R., Mishkin, A., Chien, S., Estlin, T., Backes, P., Cooper, B., Rabideau, G., and Engelhardt, B., "An Integrated Planning and Scheduling Prototype for Automated Mars Rover Command Generation," in [*European Conference on Planning (ECP 2001)*], (Sep 2001). Toledo, Spain.

[6] Arkin, R. C., Collins, T. R., and Endo, Y., "Tactical Mobile Robot Mission Specification and Execution," in [*Proc. SPIE 3838, Mobile Robots XIV*], 150–163 (Nov 1999).

[7] MacKenzie, D. C., Arkin, R., and Cameron, J. M., "Multiagent Mission Specification and Execution," *Autonomous Robots* **4**, 29–52 (Jan 1997).

[8] Endo, Y., MacKenzie, D. C., and Arkin, R. C., "Usability evaluation of high-level user assistance for robot mission specification," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **34**, 168–180 (May 2004).

[9] Ulam, P., Endo, Y., Wagner, A., and Arkin, R., "Integrated Mission Specification and Task Allocation for Robot Teams - Design and Implementation," in [*IEEE International Conference on Robotics and Automation*], 4428–4435 (Apr 2007).

[10] Lyons, D. M., Arkin, R. C., Nirmal, P., Jiang, S., Liu, T. M., and Deeb, J., "Getting it Right the First Time: Robot Mission Guarantees in the Presence of Uncertainty," in [*IEEE International Conference on Intelligent Robots and Systems*], 5292–5299 (Nov 2013).

[11] Lyons, D. M., Arkin, R. C., Nirmal, P., and Jiang, S., "Designing Autonomous Robot Missions with Performance Guarantees," in [*IEEE International Conference on Intelligent Robots and Systems (IROS)*], 2583–2590 (Oct 2012).

[12] Perdomo, E. F., Gmez, J. C., Brito, A. C. D., and Sosa, D. H., "Mission specification in underwater robotics," *Journal of Physical Agents* **4**, 25–33 (Jan 2010).

[13] Object Management Group (OMG), "Business Process Model and Notation (BPMN) Version 2.0." Technical Report, http://www.omg.org/spec/BPMN/2.0/ (Jan 2011).

[14] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., "ROS: an Open-Source Robot Operating System," in [*ICRA Workshop on Open Source Software*], (2009).

[15] Camunda.org, "The Camunda BPMN Manual." https://docs.camunda.org/manual/latest/ (Feb 2017).

[16] manual pages, P. http://spinroot.com/spin/Man/promela.html (2017). accessed: 2017-02-22.

[17] Holzmann, G., [*Spin Model Checker, the: Primer and Reference Manual*], Addison-Wesley Professional, first ed. (2003).

[18] Huth, M. and Ryan, M., [*Logic in Computer Science: Modelling and Reasoning About Systems*], Cambridge University Press, New York, NY, USA (2004).