LTL Templates for Play-Calling Supervisory Control

Thomas B. Apker[*] and Benjamin Johnson[†]
United States Naval Research Laboratory, Washington D.C.

Laura Humphrey [‡]
Air Force Research Laboratory

A Playbook allows operators to design sets of tasks, in an abstract way and before a mission begins, for a team of vehicles to perform; it also allows an operator to then call the plays during the mission execution, much like the coach of a human sports team would during a game. This paper extends this Playbook concept to include a set of assertions that must be true on each vehicle to ensure successful completion of a play, provides a means of specifying contingency plans, and translates this extended Playbook into a Linear Temporal Logic specification that can be used to synthesize a correct-by-construction controller. A demonstration of this concept is given, and its implications for autonomous systems safety and reliability are discussed.

Nomenclature

C	Boolean Command proposition
B	Boolean Behavior proposition
M	Boolean Mission Sensor proposition
H	Boolean Health Sensor proposition
R	Boolean Region proposition
Subscript	
i	Variable number
c	Implies a contingency Behavior

I. Introduction

Unmanned vehicles (UxVs) have proven to be a valuable tool for many scenarios in which manned operations are unfeasible or undesirable. Most applications to date have employed single vehicles, even though teams of autonomous vehicles could more effectively accomplish critical, dangerous tasks in environments too large or dynamic for a single platform. The reason for this is the complexity inherent in multi-vehicle command and control. In particular, current concepts of operations employ multiple human operators to monitor each vehicle due to a variety of factors including complexity, safety, and reliability, which makes coordination of multiple vehicles extremely difficult.

A consistent theme of research into human-robot interaction and interfaces has therefore been an attempt to address the complexity of multi-vehicle planning. For example, Fern and Shively[1] developed a concept they call the "Playbook" that combines multiple, pre-defined tasks into a set of plays that an operator can call during the mission, allowing the vehicles to react to certain events autonomously while the operator focuses on mission-level goals. Similarly, Duquette[2] developed the Common Mission Automation Services Interface (CMASI) to allow operators to specify abstract tasks, such as area searches or orbit points, with enough information to form the constraints needed to pose a solvable vehicle routing problem, as in Karaman and Frazzoli.[3] Draper *et al.*[4] produced a prototype multi-vehicle control station that uses CMASI tasks to define portions of a Playbook or "play-calling" interface. The work in this paper extends the Playbook and play-calling concepts further to address safety and reliability concerns.

[*]Computer Engineer, thomas.apker@nrl.navy.mil, AIAA Senior Member
[†]NRC Postdoctoral Fellow
[‡]Scientist

UxVs may become unsafe when they act on incorrect or missing information. This can include an actuator failure that changes vehicle dynamics significantly, loss of communication to a ground station, or a degraded navigation solution. Once events such as these happen, the UxV is no longer able to complete its original mission, and it requires some form of intervention to generate a new trajectory or behavior appropriate to its degraded state. Most autopilots employ some form of finite state machine to handle the worst cases, often by safely terminating the flight and returning to a default waypoint, but connecting this safety-critical software with a complex planning system or human interface presents a costly challenge for current software engineering practices.

This paper examines how correct-by-construction controllers can be used within a play-calling architecture to address the safety and reliability issues associated with multiple-UxV control with minimal engineering effort. The key contribution is the idea that a Playbook can be expressed in terms of events the vehicle can control, i.e. where it goes, what tasks it performs, and events it must react to, including operator commands, salient events, and hardware faults. We will show how to construct such a Playbook and translate it into a Linear Temporal Logic (LTL) specification. The LTL specification can then be synthesized into a controller for each vehicle using the LTL Mission Planning (LTLMoP) toolkit.[5]

II. Background and Definitions

This paper describes a complex system architecture for a team of autonomous vehicles and uses a number of technical terms that have specific meanings in the different disciplines involved in this research. For clarity, this section defines a number of these terms and explains how they are used in this work, as well as how this usage relates to similar concepts in the artificial intelligence and robotics literature.

A. Plays and Play-Calling

Play-calling as a means of supervisory control was first studied by Fern and Shively[1] as a means of increasing the level of autonomy exhibited by teams of robots engaged in cooperative tasks. Their use of abstract *plays*, designed by domain experts and called by non-expert operators, produced a substantial improvement in the performance of a simulated robot team by offloading computationally intensive processes to the automation, thus allowing the operators to focus on longer term, team-level goals.

A *play* is defined in this paper as a representation of multiple courses of action that, when enacted, will achieve a common goal. In terms of cognitive architectures for instance, plays can be formally represented as cognitive domain ontologies (CDOs)[6] that encode the desired end state (e.g., collect a set of images of a point) and any constraints (e.g., the type of image to collect) on the system that accomplishes the goal. A key feature of a play is that it can be designed off-line, before the designer has enough information to generate a complete, executable plan that could be expressed, for example, as a CMASI "AutomationRequest" in Duquette.[2] In other words, a play contains the questions to ask when forming a concrete plan, and assumes that the planning system has the means to answer them when the play is called.

B. Reactive Planning

Conventional planning assumes that accomplishing a goal requires completing a serial task sequence.[7] *Reactive planning*, on the other hand, represents a set of possible plans, often as a Finite State Automaton (FSA), that encode how to switch between tasks given specific inputs or observations. Examples of such reactive planning include bio-inspired behavior selection,[8] the physics inspired finite state machine in Martinson and Apker,[9] and the patrol allocation system in Mather and Hsieh.[10] Patzek et. al.[11] describe a mission planning and supervisory control system that uses an intuitive interface to design a reactive plan.

Representations of tasks, behaviors, and observations are unique to each of the approaches cited above, but they share the property that the actions the system can take and the observations it can sense have a Boolean representation. This suggests that, by restricting a set of plays' unanswered questions to Boolean predicates, it is possible to generate a single FSA at the beginning of a mission rather than generating a new plan every time a play is called.

C. Specification of Reactive Plans

Several tools exist to facilitate the synthesis of an FSA from a logical task specification. Two such tools are the LTLMoP toolkit[5] and Temporal Logic Planner (TuLiP).[12] The LTLMoP toolkit, which was leveraged for the work presented here, uses task specifications written in the Generalized Reactivity (1) (GR(1)) subset of LTL to produce a game that the system plays against the environment.[13] If a strategy exists for the system to win this game (by satisfying the specification), regardless of any allowed changes in the environment, that strategy can be extracted as a correct-by-construction FSA controller.

Of particular interest in this paper is the structure of a GR(1) specification and the meanings of the LTL symbols commonly used in such a specification; a more complete definition of the syntax and semantics of LTL can be found in Emerson.[14] An LTL formula of the form $\square\phi$ (read "always ϕ") states that the sub-formula ϕ holds true at all times, and a formula of the form $\square\Diamond\phi$ ("always eventually ϕ") states that the sub-formula ϕ will repeatedly become true. The formula $\bigcirc\phi$ ("next ϕ") is used to indicate that ϕ is true in the next time step. A conjunction of formulas $\phi_1 \wedge \phi_2$ ("ϕ_1 and ϕ_2") indicates that both of the formulas are true, while a disjunction $\phi_1 \vee \phi_2$ ("ϕ_1 or ϕ_2") indicates that at least one is true. Additionally, the formula $\neg\phi$ ("not ϕ") is used to indicate the negation of ϕ. Finally, the logical implication is written $\phi_1 \rightarrow \phi_2$ ("ϕ_1 implies ϕ_2"), and biconditionality is written as $\phi_1 \leftrightarrow \phi_2$ ("ϕ_1 if and only if ϕ_2").

A GR(1) specification expresses a play in terms of system *actions* and environment *sensors*, where the required behavior of the system ϕ^s is specified as a response to observed changes in the environment ϕ^e. Thus, the specification takes the form $\phi^e \rightarrow \phi^s$. Both ϕ^e and ϕ^s include three components: an initial state ϕ_i, transition restrictions ϕ_t, and goal requirements ϕ_g. The initial states $\phi_i^{e,s}$ are given as formulas over Boolean propositions and define the set of possible initial configurations for the environment and the vehicle. The transition restrictions (also called "safety conditions") $\phi_t^{e,s}$ are formulas of the form $\square\phi$, and define conditions that must hold at all times, thereby restricting the possible transitions of the system. Finally, the goal requirements (called "fairness conditions" for the environment and "liveness conditions" for the robot) $\phi_g^{e,s}$ define the configurations that the environment and robot are required to repeatedly achieve during an infinite execution.

III. Problem Statement

A conventional plan is brittle because of its serial nature. In general, an automated planner assumes it knows the state of the world, before, during and after each task. During execution, however, these assumption may prove to be untrue, and replanning after the original plan's assumptions fail is both computationally expensive and cognitively challenging for a human operator. As a result, autonomous systems are often unreliable and difficult to use. Among the most common sources of planning system failure are:

1. Hardware faults
2. Unscheduled, important observations
3. Changes in operator intentions

A *strategy* can be robust to predictable problems if it encodes a response for each possible event. However, designing a strategy is a cognitively intense and error prone task, even for experts, and difficult to do using synthesis tools based on formal methods. The approach adopted here is to specify a set of *plays* for an operator to call (addressing 3), the requirements and triggers for executing them (addressing 2), and a set of contingency plans for when things go wrong (addressing 1). The approach then translates these plays, requirements, and contingencies into an FSA that can observe both the vehicle's environment and the operator's intentions and act on them in a safe and correct manner.

The GR(1) subset of LTL allows fast synthesis of FSAs that encode a correct-by-construction strategy to allow an autonomous vehicle to accomplish its goals in an adversarial environment. The challenge that this work addresses is how to codify plays as simply as possible in a way that allows them to be translated into a GR(1) specification and synthesized into an FSA that enables an operator to direct the vehicle via commanded plays.

IV. LTL Play Calling

In a GR(1) specification, the system controls the actions it takes, as specified in ϕ^s. As such, anything that is directly controlled by the vehicle maps to a Boolean *action* proposition. As described in Kress-Gazit et al.,[15] this includes both performing a specific behavior or task, B, or going to a specified region, R. In both cases, the changes in the action propositions that result from a change in the FSA state can be used to generate a trajectory for a vehicle that advances it towards accomplishing the play's objective. The environment controls the events to which the vehicle must react (via the *sensor* propositions). This includes the operator's commands, C, i.e. the play calls themselves, as they are not under the vehicle's control. It can also include mission sensors, M, which abstractly represent the output of classifiers that detect salient features of the vehicle's environment. Finally, in order to maintain the bisimilarity between the specification and fault-prone vehicle, the Playbook includes health sensors, H, which represent the output of classifiers that observe the vehicle's own capacity to complete the play.

A. Playbook Using Boolean Propositions

To allow scalable play calling, each vehicle must have a Playbook structure that contains the information it requires to execute one or more plays. Its components are:

1. **Vehicle ID:** A unique identifier of the vehicle for which the controller is being synthesized
2. **Regions Map:** A set of LTL safety statements describing the connectivity of the physical environment
3. **Play Structures:** A list of Play Structures, see Table 1, that define each play that can be called
4. **Contingency Structures:** A list of Tasks to perform if Health Sensors become false, see Table 2

To enable the synthesis of FSAs with the capabilities and limitations of specific vehicles in mind, each vehicle is assigned a unique identifier to distinguish it from the other vehicles. By explicitly linking each template to a specific vehicle (via the Vehicle ID), the generated FSA is guaranteed to be loaded on the correct vehicle and that the FSA receives appropriate command inputs. In many cases, there will be multiple copies of the same FSA running on different vehicles with similar or identical capabilities and limitations.

In order to link the vehicle's tasks and behaviors to the physical space in which it is operating, that space is discretized into a set of mutually exclusive regions $\{R_i\}$. Movement between these regions is then modeled based on the adjacency of the regions, and formulas are added to the set of robot safety conditions ϕ_t^s that restrict the transitions of the robot's state such that the vehicle may only transition to a new state with the same region or an adjacent region. In some cases, such as an open floor or airspace, it is sufficient to connect all of the Play Structures' regions to a single "others" region, R_o. This leads to a star pattern in the connectivity graph, where a central region is connected to each other region in the graph; an example of this is given in (1) for a space with two regions of interest: R_1 and R_2. Note that the last formula in the Regions Map restricts the state space such that only one region can be *true* at a time using the exclusive-or operator $\veebar$, where $a \veebar b \equiv ((a \wedge \neg b) \vee (\neg a \wedge b))$.

$$\begin{aligned}\phi_t^s = &\Box\,(R_1 \rightarrow (\bigcirc R_1 \vee \bigcirc R_o)) \ \wedge \\ &\Box\,(R_2 \rightarrow (\bigcirc R_2 \vee \bigcirc R_o)) \ \wedge \\ &\Box\,(R_o \rightarrow (\bigcirc R_1 \vee \bigcirc R_2 \vee \bigcirc R_o)) \ \wedge \\ &\Box\,(\bigcirc R_o \veebar \bigcirc R_1 \veebar \bigcirc R_2)\end{aligned} \tag{1}$$

The specific fields that are used to specify a Play Structure and Contingency Structure are shown in Tables 1 and 2, respectively. In the case of the Play Structure, the *Command*, *Behavior 1*, and *Health Sensors* fields are required to specify a play, while the *Region 1*, *Mission Sensor*, *Behavior 2*, and *Region 2* fields are optional. The translation of a given Play Structure or Contingency Structure into a specification are discussed in sections B and C, respectively.

B. Parsing Play Structures into LTL

A Play Structure, as shown in Table 1, contains the information required to execute the command C. At a minimum, this requires a primary Task, B_1, and set of health sensors $\{H\}$, to allow the parsing tool to

Table 1. Fields of the Play Structure

Name	Symbol	Description	Units
Command	C	Senses if the Play is active	*Boolean*
Behavior 1	B_1	Defines how to accomplish the Play's primary task	*CMASI Task*
Region 1	R_1	Defines the region in which to begin the Behavior 1	*Polygon*
Mission Sensor	M	Determines when to switch between B_1 and B_2	*Boolean*
Behavior 2	B_2	Defines how to accomplish the Play's secondary task	*CMASI Task*
Region 2	R_2	Defines the region in which to begin the Behavior 2	*Polygon*
Health Sensors	$\{H\}$	Set of Health Sensors that must be true to perform the behaviors	*Boolean Array*

Table 2. Fields of the Contingency Structure

Name	Symbol	Description	Units
Health False	$\{H_f\}$	Set of Health Sensors that must be false to trigger the contingency behavior	*Boolean Array*
Health True	$\{H_t\}$	Set of Health Sensors that must be true to perform the contingency behavior	*Boolean Array*
Contingency Behavior	B_c	The behavior to perform when the listed health sensors fail	*CMASI Task*
Region	R	The region in space to perform the contingency behavior	*Polygon*

generate a pair of formulas of the form shown in (2). These LTL statements cause the synthesized FSA to perform the behavior B_1 whenever commanded to by command C, as long as the set of health sensors $\{H\}$ permit doing so (i.e., each of the Boolean sensors in the array is *true*).

$$\begin{aligned} \phi_t^s &= \Box \left(B_1 \leftrightarrow \left(C \bigwedge_{H_i \in \{H\}} H_i \right) \right) \\ \phi_g^s &= \Box\Diamond \left(\left(C \bigwedge_{H_i \in \{H\}} H_i \right) \rightarrow B_1 \right) \end{aligned} \tag{2}$$

In many cases, it helpful to restrict a behavior to a particular region, and so the Play Structure includes an optional region proposition, R_1, which augments the behavior's safety statement as shown in (3). This addition to the LTL statement requires that the vehicle first traverse the Regions Map to reach R_1, before it can perform the desired behavior B_1. This allows the FSA to handle vehicle path planning in addition to task assignment.

$$\begin{aligned} \phi_t^s &= \Box \left(B_1 \leftrightarrow \left(C \wedge R_1 \bigwedge_{H_i \in \{H\}} H_i \right) \right) \\ \phi_g^s &= \Box\Diamond \left(\left(C \bigwedge_{H_i \in \{H\}} H_i \right) \rightarrow B_1 \right) \end{aligned} \tag{3}$$

To support automated retasking when a salient event is detected, the Play Structure can contain a secondary task, B_2, second region, R_2, and mission sensor, M. This allows the parsing tool to generate formulas of the form shown in (4), where the vehicle can be commanded to perform behavior B_1 until the Mission Sensor M becomes true, at which point the vehicle will automatically switch to behavior B_2. Note that, while they are included in (4), the regions R_1 and R_2 can be omitted, if their respective tasks are not constrained to a particular region.

$$\begin{aligned}
\phi_t^s &= \square\left(B_1 \leftrightarrow \left(C \bigwedge_{H_i \in \{H\}} H_i \wedge R_1 \wedge \neg M\right)\right) \\
\phi_t^s &= \square\left(B_2 \leftrightarrow \left(C \bigwedge_{H_i \in \{H\}} H_i \wedge R_2 \wedge M\right)\right) \\
\phi_g^s &= \square\lozenge\left(\left(C \bigwedge_{H_i \in \{H\}} H_i\right) \rightarrow (B_1 \vee B_2)\right)
\end{aligned} \tag{4}$$

This formula allows the UxV to switch tasks when M changes, implicitly forming a two-state FSA associated with the nominal case of executing the play. With more complex LTL statements, one could link a single command sensor to arbitrarily complex combinations of Tasks and Mission Sensors, providing an LTL representation of the controllers described by Patzek et al.[11] However, each of these Mission Sensors is a Boolean classifier operating over noisy inputs, and is thus subject to both false negatives and false positives. Adding Mission Sensors, therefore, increases flexibility at the cost of reducing reliability.

In this paper, it is assumed that only a single Play is called at a time for a vehicle, as shown in (5), which states that if command C_i is true, then all other commands C_j are false. This substantially reduces the size of the synthesized FSA, as there are N command transitions from each state instead of 2^N.

$$\begin{aligned}
\phi_t^e = \square((\bigcirc C_1 &\rightarrow (\neg \bigcirc C_2 \wedge \ldots \wedge \neg \bigcirc C_n)) \wedge \\
&\square\,(\bigcirc C_2 \rightarrow (\neg \bigcirc C_1 \wedge \neg \bigcirc C_3 \ldots \wedge \neg \bigcirc C_n)) \wedge \\
&\vdots \\
&\square\,(\bigcirc C_n \rightarrow (\neg \bigcirc C_1 \wedge \neg \bigcirc C_2 \ldots \wedge \neg \bigcirc C_{n-1})))
\end{aligned} \tag{5}$$

To further reduce the size of the FSA, the initial state of the robot and the environment are restricted as shown in (6). Initially, the Command and Mission Sensors are all *false*, while the Health Sensors are assumed to be *true*. With all commands *false*, the vehicle will remain in place or orbit an original waypoint, depending on vehicle dynamics, until one of its Command Sensors becomes true.

$$\begin{aligned}
\phi_i^e &= \neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_n \wedge \neg M_1 \wedge \neg M_2 \wedge \ldots \wedge \neg M_n \bigwedge_{H_i \in \{H\}} H_i \\
\phi_i^s &= \neg B_1 \wedge \neg B_2 \wedge \ldots \wedge \neg B_m \wedge \neg R_1 \wedge \neg R_2 \wedge \ldots \wedge R_o
\end{aligned} \tag{6}$$

C. Parsing Contingency Structures into LTL

The use of Health Sensors in the Play Structures serves two important purposes. The first is that they prevent a vehicle from attempting to perform a behavior if it is unable to do so. The second is that they classify the remaining capability of the system and thus support contingency planning. In order to specify contingency behaviors for inclusion in the FSA, a set of *Contingency Structures* are included, in a manner similar to the Play Structures. These Contingency Structures contain the fields shown in Table 2. The set of LTL formulas generated from a contingency structure takes the form shown in (7). Because the liveness conditions generated by the Play Structures (i.e., ϕ_g^s in (2)-(4)) include the health sensors in their preconditions (the left hand side of the implication), separate liveness conditions must be generated to activate the contingency behaviors, when appropriate. The liveness condition ϕ_g^s defined in (7) requires the vehicle to attempt to perform the contingency behavior B_c when the set of Health Sensors in $\{H_f\}$ are *false* and the set of Health Sensors in $\{H_t\}$ are *true*. The safety formula ϕ_t^s in (7) restricts the contingency behavior to only be performed when all Health Sensors in $\{H_t\}$ are *true*, and when the vehicle is in region R.

$$\begin{aligned}\phi_t^s &= \Box\left(B_c \leftrightarrow \left(R \bigwedge_{H_i \in \{H_t\}} H_i\right)\right)\\ \phi_g^s &= \Box\Diamond\left(\left(\bigwedge_{H_i \in \{H_f\}} \neg H_i \bigwedge_{H_j \in \{H_t\}} H_j\right) \rightarrow B_c\right)\end{aligned} \tag{7}$$

V. Demonstration

As a demonstration of the LTL-template Playbook approach, consider an Intelligence, Surveillance, and Reconnaissance (ISR) mission scenario for one or more unmanned aerial vehicles (UAVs), as depicted in Figure 1. In this scenario, there are four regions: general airspace R_o, the area around the UAV ground control station R_{CS}, a building complex near the ground control station R_{BC}, and a forest far away from the ground control station R_F. Using an equation in the form of (1), this yields the connectivity graph shown in (8).

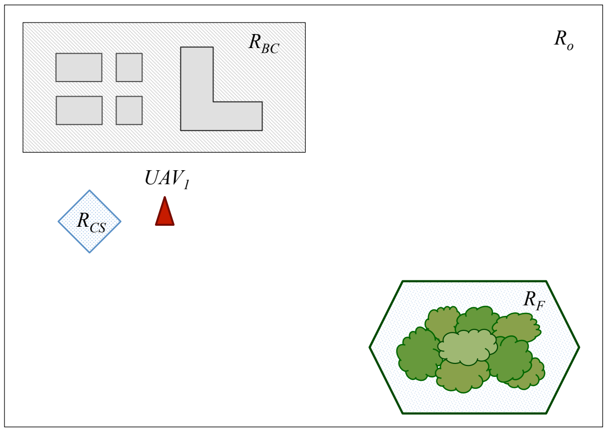

Figure 1. An example UAV ISR mission scenario with four regions and available variables as in Table 3.

$$\begin{aligned}\phi_t^s = &\Box\left(R_o \rightarrow \bigcirc R_o \vee \bigcirc R_{CS} \vee \bigcirc R_{BC} \vee \bigcirc R_F\right) \wedge\\ &\Box\left(R_F \rightarrow \bigcirc R_o \vee \bigcirc R_F\right) \wedge\\ &\Box\left(R_{CS} \rightarrow \bigcirc R_o \vee \bigcirc R_{CS}\right) \wedge\\ &\Box\left(R_{BC} \rightarrow \bigcirc R_o \vee \bigcirc R_{BC}\right) \wedge\\ &\Box\left(\bigcirc R_o \veebar \bigcirc R_F \veebar \bigcirc R_{CS} \veebar \bigcirc R_{BC}\right)\end{aligned} \tag{8}$$

Since the building complex R_{BC} is close to the control station R_{CS}, the scenario assumes that the UAVs surveilling R_{BC} are able to use a high-bandwidth data link to stream imagery from their sensors back to the control station R_{CS}. However, since the forest R_F is not as close, UAVs surveilling this area must store imagery from their sensors using onboard memory, then fly back to the control station R_{CS} to upload the collected imagery for human viewing. For behaviors, the UAVs have atomic tasks available for searching the entirety of an area, searching the perimeter around an area, automatically tracking a target that has been detected, uploading stored data through a wireless connection to a receiving station, and landing at a specified location to refuel.

To construct plays for the Playbook, the behaviors, sensors, and regions listed in Table 3 are used in the templates given in Tables 1 and 2. Command variables for each play are chosen by the play designer when creating a new play. For contingencies in this scenario, the primary concern will be the UAVs running out of fuel. To address this concern, the parameters for the Contingency Structure given in Table 2 are set to engage in behavior B_{Refuel}, in region R_{CS}, when sensor S_{Fuel} is false. The full parameter list for the Contingency Structure is given in Table 4.

Table 3. Available Template Variables

Behaviors		Sensors		Regions
$B_{SearchArea}$	Fly a trajectory that covers a region	S_{Target}	The UAV has detected a ground target	R_o
$B_{SearchPerimeter}$	Fly a trajectory on the border of a region	$S_{DataOut}$	The UAV has no available data storage capacity	R_{CS}
$B_{TrackTarget}$	Track a detected ground target	S_{Fuel}	The UAV has sufficient fuel to continue	R_{BC}
$B_{DeliverData}$	Send or offload stored data	S_{Sensor}	The UAVs sensor is functioning	R_F
B_{Refuel}	Refuel the UAV			

Table 4. The Contingency Structure for the Mission depicted in Figure 1

Health False	Health True	Contingency Behavior	Region
$\{S_{Fuel}\}$	$\varnothing$	B_{Refuel}	R_{CS}

With the Contingency Structure defined, suppose the play designer wants a play called $TrackTargetBC$ to search for a ground target in R_{BC} and track it if it is found. The play designer then sets the play parameters as in Table 5, with the primary behavior as $B_{SearchArea}$ and the secondary behavior as $B_{TrackTarget}$. The secondary behavior is activated when a target is found, as indicated by the sensor S_{Target}. Both behaviors are enabled in region R_{BC} and require that the UAV have sufficient fuel and a functioning sensor, so the Health Sensors are set to $\{S_{Fuel}, S_{Sensor}\}$. The reactive nature of this play is demonstrated in three different hypothetical situations shown in Figure 2, Figure 3, and Figure 4, with changes to behaviors and sensor propositions as labeled. In each of these figures, the UAV starts in R_o near R_{CS} with behavior $B_{SearchArea}$, and sensor propositions S_{Fuel} and S_{Sensor} set to $true$ and S_{Target} set to $false$. Over the period of time depicted in Figure 2, the UAV searches R_{BC} but never finds a target or runs out of fuel, so its behavior never changes. In Figure 3, the UAV finds a target in R_{BC}, causing sensor S_{Target} to become $true$, enabling behavior $B_{TrackTarget}$. And in Figure 4, the UAV finds the target, causing S_{Target} to become $true$ and behavior $B_{TrackTarget}$ to be enabled; however, it then runs out of fuel so that S_{Fuel} becomes $false$, and behavior B_{Refuel} is enabled instead. As the UAV goes to refuel, it loses the target and S_{Target} also becomes $false$. When the UAV refuels, S_{Fuel} becomes $true$ again, and behavior $B_{SearchArea}$ is enabled again.

Suppose the play designer also wants a play called $SearchForest$, to search the forest in region R_F. Recall that a key difference between regions R_{BC} and R_F is that imagery from the UAV can be streamed back through a live feed from R_{BC} but not from R_F. Therefore, the UAV will need to return with its collected imagery from R_F so it can be viewed by a human operator. Anticipating this, the play designer then sets the play parameters as in Table 6, with the primary behavior as $B_{SearchArea}$ in R_F, the secondary behavior as $B_{DeliverData}$ in R_{CS}, and the Mission Sensor as $S_{DataOut}$. As with $TrackTarget_{BC}$, behaviors require the UAV have sufficient fuel and a functioning sensor, so the Health Sensors are set to $\{S_{Fuel}, S_{Sensor}\}$.

Now suppose that the play designer is concerned about surveilling the perimeter of the forest rather than its interior. The play designer could then create a play $SearchForestPerimeter$ that is the same as $SearchForest$, with the simple change of setting the primary behavior to $B_{SearchPerimeter}$. The parameters for this play are as given in Table 7. Figure 5 shows these two plays, in particular how simple switching

Table 5. A play $TrackTargetBC$ to search for and track a target in R_{BC}

Command	Behavior 1	Region 1	Mission Sensor	Behavior 2	Region 2	Health Sensors
$C_{TrackTargetBC}$	$B_{SearchArea}$	R_{BC}	S_{Target}	$B_{TrackTarget}$	R_{BC}	$\{S_{Fuel}, S_{Sensor}\}$

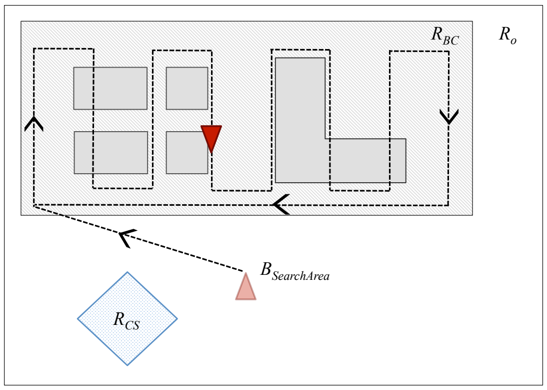

Figure 2. The play $TrackTargetBC$, assuming no targets are found and fuel remains sufficient

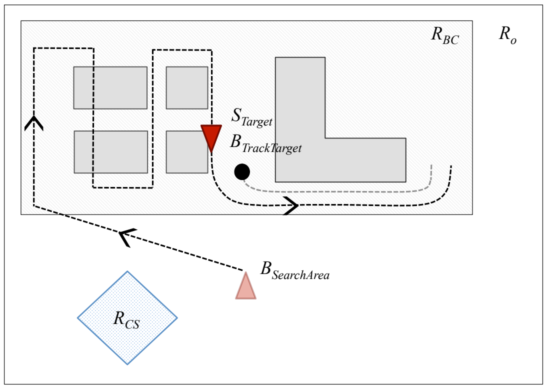

Figure 3. The play $TrackTargetBC$, assuming a target is found and fuel remains sufficient

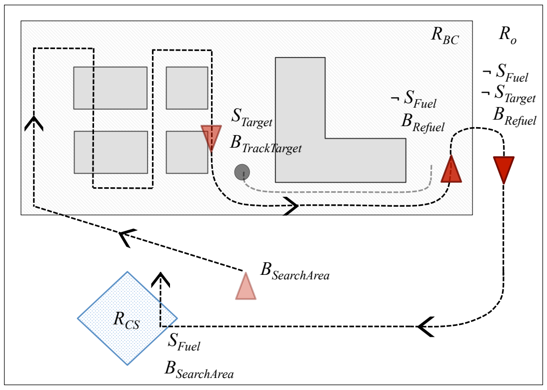

Figure 4. The play $TrackTargetBC$, assuming a target is found but then fuel is no longer sufficient

between the two plays is achieved by first enabling the command variable $C_{SearchForest}$, then later enabling the command $C_{SearchForestPerimeter}$. In either case, the UAV will return to R_{CS} if either S_{Fuel} or S_{Data} become $false$, enabling behaviors B_{Refuel} and $B_{DeliverData}$, respectively.

Table 6. A play ***SearchForest*** **to surveil and return with imagery of** R_F

Command	Behavior 1	Region 1	Mission Sensor	Behavior 2	Region 2	Health Sensors
$C_{SearchForest}$	$B_{SearchArea}$	R_F	$S_{DataOut}$	$B_{DeliverData}$	R_{CS}	$\{S_{Fuel}, S_{Sensor}\}$

Table 7. A play ***SearchForestPerimeter*** **to surveil and return with imagery of the perimeter of** R_F

Command	Behavior 1	Region 1	Mission Sensor	Behavior 2	Region 2	Health Sensors
$C_{SearchForestPerimeter}$	$B_{SearchPerimeter}$	R_F	$\neg S_{Data}$	$B_{DeliverData}$	R_{CS}	$\{S_{Fuel}, S_{Sensor}\}$

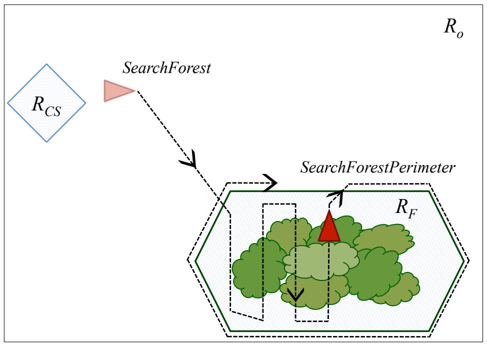

Figure 5. Switching between the plays ***SearchForest*** **and** ***SearchForestPerimeter***

VI. Conclusion

The original Playbook concept provided a means for an operator to design tasks for vehicles offline, when there is time to reason over what a vehicle *should* do in various circumstances. This extension of the Playbook architecture gives the operator the ability to specify what must be true to perform those tasks, as well as provide contingent plans when those assertions fail. By abstractly representing both the nominal tasks and their requirements as Boolean propositions, we can translate this Playbook into an LTL specification that produces a correct-by-construction FSA which permits both a simple command interface and plan monitoring with automatic replanning.

With this system, the complex problem of designing safe and reliable UxV controllers can be reduced to determining the requirements of each play and implementing classifiers that concretize the Boolean abstraction used by the FSA. Assuming each portion of the controller (i.e. the guidance system, FSA implementation and the classifiers) cannot affect the others' internal state, the components can be tested independently and in narrowly defined state spaces. This addresses a major problem in the verification and validation of autonomous systems, in addition to improving their operational utility.

Future research will investigate automating the generation of Health Sensors given the trajectory planner and guidance systems that will be used to implement each behavior or task, as well as investigating the impact of uncertain classifiers and how to mitigate it.

Acknowledgments

This work was performed at the Naval Research Laboratory. The authors were funded by the Office of Strategic Defense.

References

[1]Fern, L. and Shively, R. J., "A comparison of varying levels of automation on the supervisory control of multiple UASs," *Proceedings of AUVSIs Unmanned Systems North America 2009*, 2009, pp. 10–13.

[2]Duquette, M., "The Common Mission Automation Services Interface," *InfoTech at Aerospace*, 2011.

[3]Karaman, S. and Frazzoli, E., "Vehicle Routing with Temporal Logic Specifications: Applications to Multi-UAV Mission Planning," *Journal of Robust and Nonlinear Control*, Vol. 21, 2011, pp. 1372–1395.

[4]Draper, M., Miller, C. A., Benton, J., Calhoun, G. L., Ruff, H., Hamell, J., and Barry, T., "Multi-Unmanned Aerial Vehicle Systems Control via Flexible Levels of Interaction: An Adaptable Operator-Automation Interface Concept Demonstration," *AIAA Infotech@Aerospace (I@A) Conference*, 2013.

[5]Finucane, C., Gangyuan Jing, and Kress-Gazit, H., "LTLMoP: Experimenting with language, Temporal Logic and robot control," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, IEEE, Piscataway, NJ, USA, 2010 2010, pp. 1988–93, 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), 18-22 Oct. 2010, Taipei, Taiwan.

[6]Douglass, S. and Mittal, S., "A Framework for Modeling and Simulation of the Artificial," *Ontology, Epistemology, and Teleology for Modeling and Simulation*, edited by A. Tolk, Vol. 44 of *Intelligent Systems Reference Library*, Springer Berlin Heidelberg, 2013, pp. 271–317.

[7]Smith, D., Frank, J., and Jonsson, A., "Bridging the gap between planning and scheduling," *KNOWLEDGE ENGINEERING REVIEW*, Vol. 15, No. 1, MAR 2000, pp. 47–83.

[8]Dasgupta, P., "Multi-agent coordination techniques for multi-robot task allocation and multi-robot area coverage," *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, May 2012, pp. 75–75.

[9]Martinson, E. and Apker, T., "A Physics Inspired Finite State Machine Controller for Mobile Acoustic Arrays," *Distributed Autonomous Robotic Systems, 11th Edition*, 2014.

[10]Mather, T. W. and Hsieh, M. A., "Macroscopic Modeling of Stochastic Deployment Policies with Time Delays for Robot Ensembles," *ijrr*, 2011.

[11]Patzek, M., Rothwell, C., Bearden, G., Ausdenmoore, B., and Rowe, A., "SUPERVISORY CONTROL STATE DIAGRAMS TO DEPICT AUTONOMOUS ACTIVITY," Tech. Rep. 1234567, DTIC, 2013.

[12]Wongpiromsarn, T., Topcu, U., and Murray, R., "Receding horizon temporal logic planning for dynamical systems," *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, Dec 2009, pp. 5997–6004.

[13]Piterman, N., Pnueli, A., and Sa'ar, Y., "Synthesis of Reactive (1) Designs," *Verification, Model Checking, and Abstract*, 2006, pp. 364–380.

[14]Emerson, E. A., "Temporal and Modal Logic," *Handbook of Theoretical Computer Science*, Vol. E, No. 16, July 1990, pp. 995–1072.

[15]Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J., "Temporal Logic based Reactive Mission and Motion Planning," *IEEE Transactions on Robotics*, Vol. 25, No. 6, 2009, pp. 1370–1381.