A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles

Christopher A. Miller, Chief Scientist
Robert P. Goldman, Senior Scientist
Harry B. Funk, VP Research
Peggy Wu, Scientist
Smart Information Flow Technologies
Minneapolis, MN.
{cmiller, hbfunk, rpgoldman, pwu}@SIFT.info

Billy B. Pate
Systems Engineer
Geneva Aerospace
Addison, TX
bpate@genevaaerospace.com

Abstract

As Unmanned Air Vehicles (UAVs) become more common, several core challenges emerge including the need for single operators to control, or perhaps more accurately, to request services from, multiple UAVs without the degree of workload, specialized training and platform-specific control interfaces which characteristize of current UAV operation. A challenging problem involves the dismounted, small unit soldier who may need services from a range of vehicles while in challenging situations such as urban combat. We are developing an approach to UAV control that addresses these challenges. We use the metaphor of a sports team's playbook to enable both quick and complex, variable-initiative control of a variety of UAVs—the user of our Playbook "calls a play" which need not (but may, if the user desires) specify what type or how many UAVs should be used to satisfy it. We describe the Playbook-enhanced Variable Autonomy Control System (PVACS), which combines SIFT's Playbook architecture and Geneva Aerospace's Variable Autonomy Control System (VACS). We illustrate the capabilities and operations of PVACS in an extended walkthrough example involving a request for *Overwatch*—continued surveillance—of an urban intersection and the development and execution of a plan to satisfy that request using multiple, heterogeneous UAVs.

Introduction

As Unmanned Air Vehicles (UAVs) become more common in planning, procurement and use, several core challenges emerge. First, current ratios of multiple operators per vehicle will be unacceptable the near future. Second, current practices of providing a dedicated workstation for each vehicle or vehicle type will also be unacceptable when individuals must interact with multiple, heterogeneous vehicles. Third, as UAVs become more available to lower echelons, new usability and training requirements will be imposed. Not all operators will spend months training to be rated for a vehicle, nor will they devote full attention to vehicle management. Instead, UAVs must be controllable with much less training and while concurrently engaged in many other activities—even taking fire.

The authors are developing an approach to UAV control that will address the above challenges. We use the metaphor of a sports team's playbook to enable both quick and complex variable-initiative control with a variety of UAVs—the user of our Playbook "calls a play" which is akin to requesting a service from one or many UAVs which may or may not be specified in the request. We are testing this approach in the Playbook-enhanced Variable Autonomy Control System (PVACS) project, so-called because it combines SIFT's prior work on Playbook interfaces and Geneva Aerospace's prior work on the Variable Autonomy Control System (VACS).

Presented at the American Helicopter Society 60th Annual Forum, Baltimore, MD, June 7-10, 2004.

Delegation via Playbooks

We begin with the realization that the problem of effectively making use of intelligent, semi-autonomous agents to satisfy one's needs and desires, without undue added workload or attentional demands, is hardly a new one in human history. In fact, human supervisors of human subordinates have been solving this problem for millennia. Humans generally don't control other human agents at the level of individual movements—roughly analogous to the current state of affairs where UAVs must be controlled via joystick commands—and in fact, there are strong reasons why this is never an effective management strategy (e.g., [1]). Instead, human supervisors delegate tasks to subordinates, or they request services that subordinates determine how best to satisfy. In either case, some objective or partial plan is communicated to the subordinate, perhaps along with constraints on how that objective may be achieved, but simultaneously some authority/autonomy for exactly how to achieve that objective in context is also given to the subordinate.

When interaction with subordinates is effective it is because the effort the supervisor requires to request, instruct, oversee and manage the subordinate(s) is less than s/he

would require to do the task without them, and/or because more tasks can be accomplished faster or better via delegation than would otherwise be the case. This saving or extending of the supervisor or requestor's effort comes about specifically because the s/he does not have to plan and execute every detail of the desired behavior at the lowest levels of control available in the system. Generally, more efficiency will be achieved the greater the degree of responsibility for planning and execution the requester can *reliably* delegate to his/her subordinates. Where the reliability of delegation (in terms of having the desired outcome achieved via a desired method) breaks down, supervisors can actually incur greater workload than if they had done the task themselves—because not only did they have to instruct and monitor the subordinate in the first place, but now they have to repair errors as well.

Domains with a long history of attention to how to communicate intent effectively while simultaneously minimizing supervisor workload include business, construction, military command and control and sports teams. The latter two are of special interest to us for the designing of a control architecture for UAVs. Many sports teams (particularly, but not exclusively, American football teams) achieve complex forms of coordinated behavior by means of a "playbook". A playbook contains predefined patterns of behavior that are understood by all participants on the team. By means of a single, short play name or label, the quarterback or team captain can express his/her intent for a large number of independent actors to behave in a dynamically-changing yet coordinated, focused, constrained and effective fashion. Furthermore, plays can serve as a shared point of reference from which to build novel variations with minimal effort. But it is worth noting that plays also require that the actors be capable of interpreting and applying the play to the context that exists when the play is called. This may be as simple as deciding whether to step left or right depending on what direction the opponent is coming from in a football play, but it precludes completely rote behavior. Actors must be allowed some autonomy about how to perform their delegated roles if there is to be any efficiency gain in the system.

Service requests are like play calling in all but the degree of authority wielded by the requestor. A supervisor is delegating tasks or goals to subordinates who are, presumably, under his or her control and have no other supervisor concurrently tasking them. A requester of service may use the same vocabulary of goals and tasks, but since his/her "commands" are not going to subordinates that are under his/her full and exclusive control, the requester has somewhat less authority than the true supervisor—and somewhat less guarantee that the request will be satisfied completely. This tradeoff may be useful, however, especially if (as is the case for many proposed battlefield UAVs) the subordinates are in high demand and can be used more effectively overall if they are shared among multiple users.

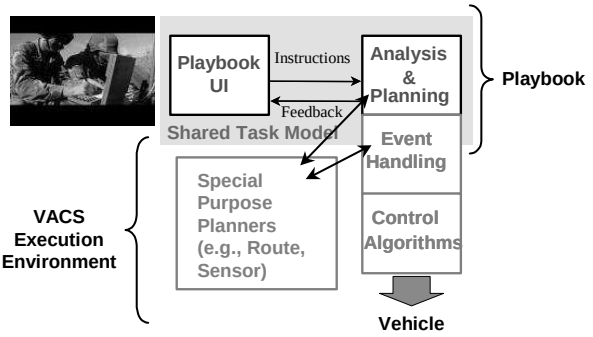

Figure 1. General Playbook Architecture.

A Playbook Architecture

We have been developing a "playbook" approach to the control of UAVs that strives to build the same kind of relationship between them and a human supervisor as exists between a captain and his/her team. Figure 1 presents the basic architecture for our Playbook. We will provide a brief discussion of the Playbook architecture here, followed by a detailed walkthrough of a specific usage example designed to illustrate both the user's experience and the internal representation and reasoning of the Playbook in the remainder of the paper.

Playbook consists of a user interface (UI) and an Artificially Intelligent Analysis and Planning Component (APC) that communicate via a shared model of the tasks that can be performed in a domain. This task model is both hierarchically and sequentially organized allowing the stringing together of tasks or "plays" in commandable sequences and/or drilling down within a given play to select alternate performance methods.

Operators can interact with and request services from automation in highly sophisticated and flexible ways via Playbook. Like the quarterback of a football team, a PVACS operator can command a very complex, very high level "play"—even one involving a heterogeneous mix of actors (vehicles)—via a fast and simple action. Also like the quarterback, the operator can issue more specific requests about individuals' behaviors, albeit spending more time to "flesh out" the request, providing more instructions about specifically how it should be satisfied.

The Playbook Analysis and Planning Component (APC) evaluates the feasibility of alternate methods of satisfying requested plays. When given a high-level play request, the APC selects among various feasible methods, issues instructions to the VACS execution environment (see below) and monitors for necessary revisions during performance. When given lower-level, more specific and detailed requests (such as a specific platform to use, a specific paths to be followed, or specific scan patterns to use), the APC reviews them for feasibility and either (a) reports when requested actions are infeasible, (b) passes 'validated' user-requested plans to the execution environment and monitors their performance, or (c) fleshes out operator high level operator requests (like "Overwatch") to an executable level

(for example, choosing among available platforms, selecting waypoints for ingress and egress, identifying sensor steering parameters, etc.) within the constraints the operator has imposed.

More details about the Playbook architecture and various UI designs are provided in [2]. In previous work, we have developed prototype playbooks for UCAV teams [3], Tactical Mobile Robots [4], and the RoboFlag (robotic capture the flag) game [5]. The remainder of this paper will focus primarily on our current adaptation of the Playbook architecture to serve as a platform for controlling, and/or requesting services from, multiple heterogeneous UAVs in an urban combat context.

VACS Control Execution Environment

Playbook by itself is an environment for human interaction and planning and does not include the event handling and control capabilities necessary to execute missions on real (or realistically simulated) vehicles. As in Figure 1, Playbook must be integrated with a control architecture that provides these capabilities. Geneva Aerospace's Variable Autonomy Control System (VACS) provides a robust integrated control architecture enabling a single operator to control multiple UAVs [6].

The VACS architecture includes an Internet Protocol (IP) based network-centric communications package linking teams of UAVs and remote operator workstations. VACS fuses sensor and database information to autonomously adjust flight profiles to avoid terrain and mid-air collisions. Once VACS adjusts the flight profile, however, it relies on the human operator to decide if the mission plan must be revised (e.g., perhaps because time on target constraints have been violated). Further, the human user must make all mission level decisions and interact with the various control levels.

More details about VACS are provided in a companion paper in this volume [7]. Integration with Playbook advances VACS to higher levels of autonomy by providing automated means of developing and adjusting plans to achieve mission objectives. Playbook possesses a hierarchical understanding of the operational intent and specific target tasking, and can provide high-level commands to the vehicle and sensor control systems following the command structure already in place in the VACS. In essence, VACS provides a "library" of control execution behaviors from which plays, as well as more complex sequences of plays which form overall mission plans, can be composed. The integrated Playbook + VACS (PVACS) capabilities are particularly relevant to projected operational concepts where busy and/or non-rated operators must supervise teams of heterogeneous vehicles. PVACS' combination of very high level and variable autonomy control will allow busy operators to command sophisticated, coordinated behaviors simply and rapidly and/or allow operators with more time or training to impose highly specific commands to customize vehicle behavior to their exact needs.

Extended Walkthrough Example: PVACS Control of Heterogeneous UAVs

PVACS is being developed to enable a small unit soldier, with minimal training, to control or request services from multiple, heterogeneous UAVs while concurrently engaged in urban combat operations. An example of PVACS operations and functionality can be provided through an extended example scenario involving variable initiative, play-like control of multiple, coordinated, heterogeneous UAVs. Our demonstration scenario involves a platoon commander engaged in urban operations who must coordinate multiple UAVs for sustained surveillance of a fixed location (e.g., an intersection) while simultaneously securing nearby buildings. Since the soldier cannot devote sustained attention to managing the UAVs, they must operate largely autonomously. Furthermore, the soldier might have little time to convey his or her intentions. S/he can task the UAV team through PVACS by "calling" a single, simple *Overwatch* play and providing a single parameter (the target area)—this constitutes a requested service which PVACS understands and will endeavor to satisfy through the best mix of vehicles available. With our UI, this series of actions will be performable in under 15 seconds.

Play Selection and Available Plays

From the operator's perspective, play calling can be an extremely simple, straightforward activity. Not only does the platoon leader in our example not need extensive training to be "rated" for control of a specific UAV, s/he may not know or care what kind of UAV will be used to satisfy the service s/he is requesting. Instead, s/he will need training only in the specific UI available to call "plays" and in the meaning of the various plays available to be called. This process will be further simplified by making use, where possible, of existing or intuitive human terminology for "plays" to be executed.

A "play" is simply a named function or behavior for one or more UAVs to perform. As such, plays may exist at finer and coarser levels—and more complex plays may be composed from simpler plays. The job of determining which degrees of freedom in play calling and play composition should be put in the hands of the human user vs. under the (sole) control of Playbook's Analysis and Planning Component (APC) is the job of a system designer. There will be some plays that can only be activated by the human user, some that require human authorization, and (frequently) some behaviors that exist at a level below that which the human has any interest, need or ability to control (e.g., low-level steering commands, fuel management behaviors, etc.) We will have more to say about the representation of plays below; for now, however, it is sufficient to regard a play as a high level behavior or action template that (a) can be requested by a simple command from a human user, (b) defines (or, perhaps more accurately, can be satisfied by) a range of possible behaviors from the subordinate entities which will enact it, (c) decisions about how, exactly, to perform the play in the current context can, but do not have to be, constrained or actively made by the hu-

man user, but (d) any decisions which are not made by the human will be made by some mix of the APC and the execution environment.

The set of plays that exist in any user's Playbook must be determined by a designer and/or by the user (or his/her supervisors) prior to use. Top-level plays are the highest, or coarsest level at which the user will be able to request services and, by virtue of being coarsest, they also represent the lowest workload (if not the most precise) way for a user to ask for a complex behavior. Examples of top-level plays (emphasizing current and near-future movement and sensor capabilities of small unit UAVs) that we have explored for use by soldiers in this urban combat setting include:

- *Overwatch*—sustained surveillance of a fixed target or area.
- *Track Target*—sustained surveillance of a moving target
- *Area Recon*—one-pass reconnaissance of an area
- *Route Recon*—one pass reconnaissance of a route
- *Watch Perimeter*—sustained surveillance of the perimeter of an area
- *Encircle Patrol*—circle a target (such as a building) while maintaining a focus inward toward the center of the area (and, hence, the exits from the building)
- *Protect*—surveillance of an area relative to my position as I move

In our concept of operations, a Playbook operator will be able to "call" any of these plays via a simple, small, portable and generic UI device such as a hardened PDA or laptop. Play calling, at this top level, will generally be restricted to selecting the desired play from a short list of alternatives and, perhaps, providing a very limited number of user input parameters. For example, to call *Overwatch* via a PDA-based UI such as the one illustrated in Figure 2, the user would select *Overwatch* from the pull-down menu on the first screen. The *Overwatch* play allows the user to stipulate many different parameters in order to customize how the play is to be performed, but only one such parameter is *required*—the target point or area for which surveillance is to be provided. This can be provided by reference to a map- or image-based display on the same device, as illustrated in the second screen in Figure 2.

Other parameters that are critical to the performance of *Overwatch* may also be available at a top level, as shown in the third screen in Figure 2. Smart default values (perhaps configurable in a pre-mission briefing) should serve to minimize the user's need to reset these variables, but they will always be available to him or her to enable a better "tuned" version of the play. Such parameters are roughly equivalent to a quarterback's ability to "call signals"—present variants to a basic play template that are quickly and easily activated, if needed, but can otherwise be ignored. Examples of such parameters for the *Overwatch* play include:

- Earliest acceptable "time on target"—or, more specifically for this play, time when *Overwatch* is established and imagery is provided.
- Earliest acceptable time off target
- Latest acceptable time on target
- Latest acceptable time off target
- Stealth—to what degree should the resulting plan strive to keep the UAV(s) used covert?
- Priority—what is the user's perceived priority for this request?
- Recipient—who should the output of this request be sent to (in addition to the requester)?
- Platform/Vehicle—which of various possible UAVs should be used for this request
- Sensor—which of various possible types of sensors should be used for this request

Note that while we can currently capture user inputs for all of these parameters, they are not yet all used in PVACS' reasoning. Current reasoning capabilities center on the various time and target parameters and the type of vehicle specified. Future work with more sophisticated sensor and maneuver models may also include sensor resolution, image distance and angle, etc., as user-settable parameters though we have not developed these capabilities as yet.

For each of these parameters, the user may input a desired value. If s/he does input a value, the value can be further designated as a "hard" or "soft" constraint. A hard constraint is one that Playbook's APC must meet if it is to have a valid plan. If no plan can be found which satisfies this constraint, Playbook must report failure to the user. Soft constraints, by contrast, are simply "nice to haves". Playbook will try to develop a plan that meets them, but retains the freedom to create plans which violate these constraints if necessary. Furthermore, Playbook contains intelligent default values for each of these parameters that are designed to frequently meet or be satisfactory to the user's needs, thereby further reducing the need for user inputs.

For example, the earliest acceptable time on target will generally be "ASAP"—a value that will be acceptable for a wide variety of instances where this play is called. On the other hand, a reasonable default for the earliest acceptable time off target would be more likely to be dependent on the specific platforms and activities planned for a given unit and mission—and should probably be tuned as a part of a pre-mission briefing. Still other defaults may be dynamically composed based on intelligent heuristics present in the Playbook. For example, the acceptable range of sensors for use in the *Overwatch* play will be different if the play is commanded during the day than if it is commanded at night.

Figure 2. Conceptual PDA-based UI for Playbook Interactions by a Small-Unit Soldier involved in Urban Combat.

Thus, for this example walkthrough, the user may request an *Overwatch* play to be performed for a designated area (e.g., an urban intersection) in as little as two user actions: (1) select *Overwatch* from the pull down play menu, and (2) designate the target area for the *Overwatch* on an associated map screen. If the operator has additional time and/or wishes to constrain the ways in which PVACS can satisfy the play, s/he may impose increasingly more detailed restrictions—for example, stipulating the type of vehicle or sensor to be used, the duration etc., by stipulating and/or overriding the default parameters that come with the baseline version of the play. In principle, but likely inappropriate for the simple PDA UI intended for field use as illustrated in Figure 2, still more detailed commanding is possible by "drilling down" into the definition of the play itself and providing restrictions as to how Playbook can choose to satisfy the play requested. To discuss this in depth, however, it will be necessary to provide some information about the representation of a play within Playbook and about how playbook makes selections among available alternatives. This will be presented in the next section below.

Play Representation and Navigation

Figure 3 provides a decomposition of the *Overwatch* play as it is represented in PVACS' task representation. All plays that PVACS knows about consist of a high-level task decomposition containing all the various known ways in which the task/play can be accomplished. For example, the *Overwatch* play, as represented in Figure 3, must contain at least one sub-play called "Overwatch Sortie". The significance of the diamond following Overwatch Sortie is that, after performing an Overwatch Sortie, the parent play (*Overwatch*) may iterate back through one or more subsequent instances of *Overwatch* Sortie, or it may not—a single instance may be sufficient to complete the play.

At any rate, drilling down further in the representation of *Overwatch*, we see that each instance of Overwatch Sortie may require an initial step of Obtain Aircraft (a task that can be satisfied by various methods including, perhaps: Requisition Loitering Aircraft, Request Launch, and Requisition Engaged Aircraft, etc.). Next, an Ingress task may be necessary and, if so, it will be composed of one or more Fly-To waypoint legs. After Ingress is complete for the vehicle associated with this Sortie, the vehicle will Scan and may also maneuver, and these actions may repeat until some condition (generally, the time requirements for the parent *Overwatch* task) are completed. Following Maneuvering and Scanning, the Sortie includes an Egress and a Destage task.

Note that the user never sees (at least not during operations) this detailed view of the plan to be executed. Instead, his or her interactions are with a simplified UI as illustrated in Figure 2 above. Nevertheless, the meaning of the user's actions has significance within the conceptual task decomposition described above. More detailed interactions would be possible via a larger format device used with more time and under less stressful conditions—for example, a laptop-based mission planning environment, perhaps integrated with FalconView,as illustrated in Figure 4.

Note also that new methods of achieving a play may be readily incorporated under this heading as they become available. If, for example, an unmanned ground vehicle were available to provide surveillance of a (limited) ground area, this capability could be represented as, simply, an alternate branch or method of performing an Overwatch Sortie task and the Playbook Analysis and Planning Com-

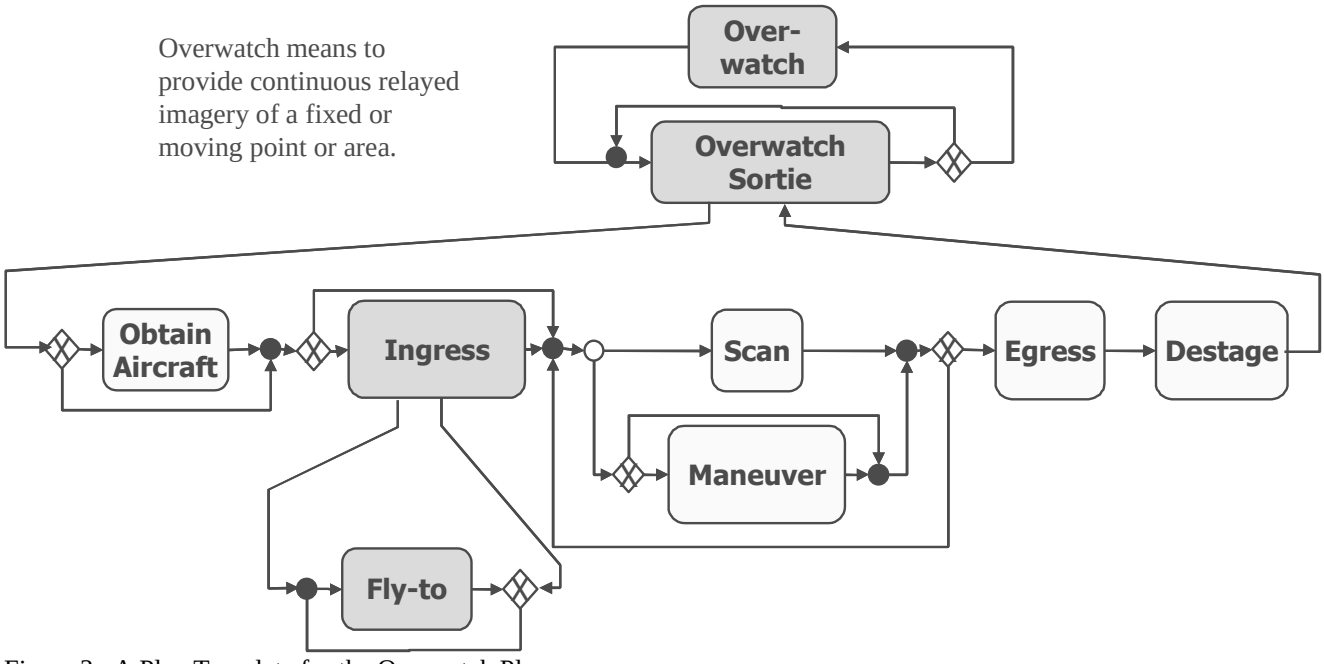

Figure 3. A Plan Template for the Overwatch Play.

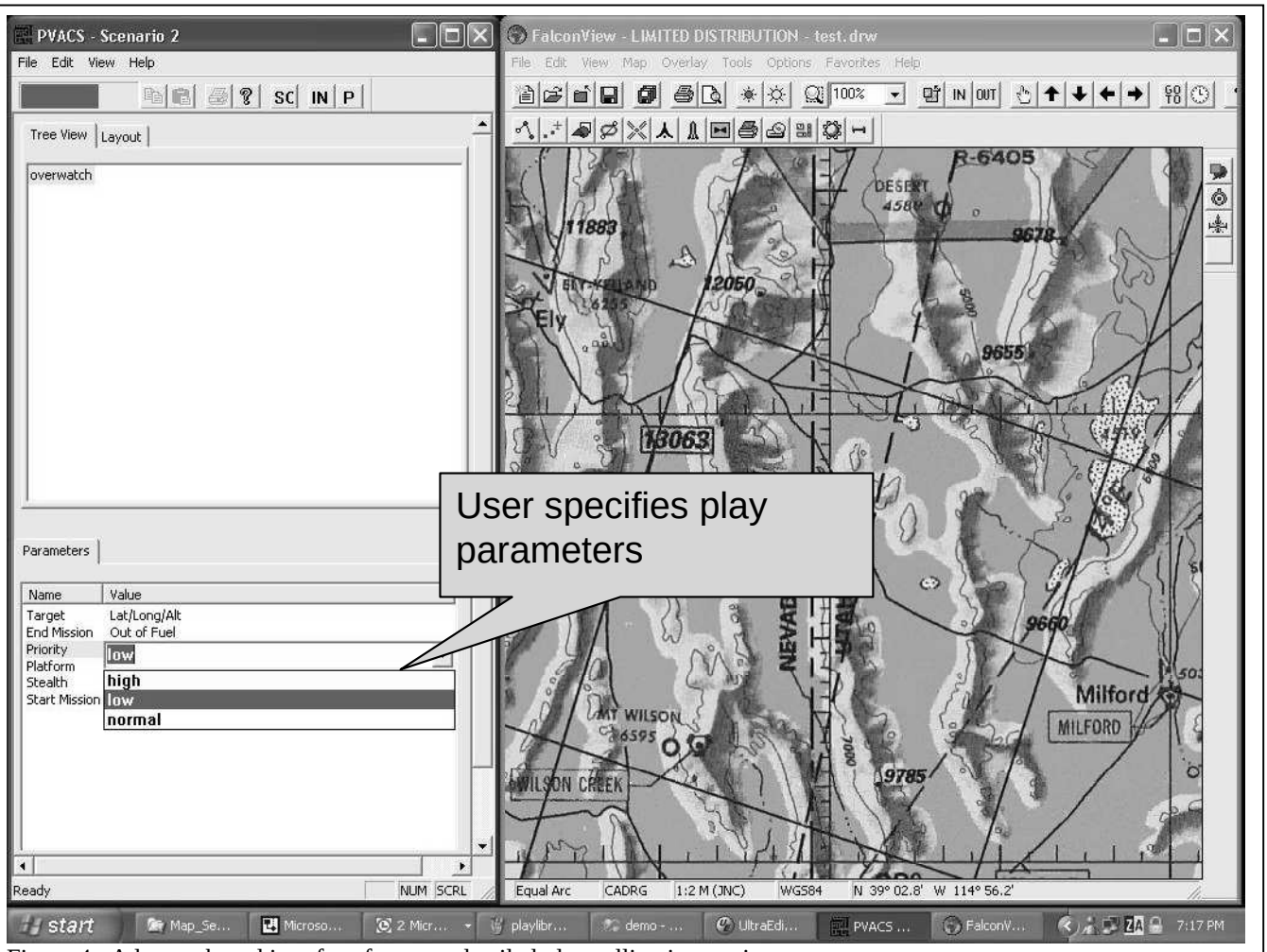

Figure 4. A laptop-based interface for more detailed play calling interactions.

ponent (APC) would reason over that branch in its attempt to develop a viable plan (as discussed below).

This representation defines what "Overwatch" means to the Playbook, the APC and, with minimal training, the human operator. When the operator "calls" an *Overwatch* play, s/he is telling the APC of Playbook that s/he wants to develop a plan for the use of UAV resources that obeys the format of the template outlined in Figure 3. Furthermore, when the operator provides additional constraints, s/he is telling the APC more about exactly which of the many possible plans which could satisfy the template are, in fact, desired.

For example, when the operator stipulates the target for the *Overwatch* play as a specific intersection of streets for a specific period of time, Playbook's APC knows that it must seek one or more sorties with available vehicles which are capable of arriving at that intersection by the latest acceptable begin time, of remaining on target until the earliest acceptable end time, and of providing imagery of the target area to the requesting operator. Furthermore, the times and positions for the Scan sub-tasks impose constraints that are propagated to the remaining portions of the task decomposition, imposing in turn, constraints on the ingress and egress routes, and even on what vehicles may be chosen. In other words, whenever the user imposes a constraint on the range of acceptable plans, s/he reduces the degrees of freedom that the Playbook has for satisfying his or her request. This will frequently be important to ensure that the right kind of *Overwatch* is provided—the kind that best meets the user's needs. It also serves as additional reasons for the user to avoid over constraining the play—because it may result in a situation where no achievable versions of the play meet the user's request.

Our goal is to provide a tasking environment in which the user may request a play or service *without* explicit regard to a specific, available UAV platform or a specific sensor type. This both minimizes the users' workload and training requirements and maximizes the ability to use available resources to meet ongoing users' needs. While users will have the opportunity to request a specific vehicle by class, equipment type or even specific tail number, they will not need to do so.

There is another reason for avoiding equipment-based requests. We argue that when the user requests a specific vehicle type, s/he is short circuiting a process (perhaps out of convenience or need) whereby the bounding characteristics of a range of acceptable solutions are provided. When the user says "I want a rotary winged UAV" there will generally be reasons for that request (e.g., s/he needs a stable image from a constant position). By stating the conclusion

rather than the motivation, s/he constrains Playbook's reasoning more than is frequently necessary (e.g., there may be fixed wing UAVs equipped with point-synchronized cameras that would provide a stable image). A request that stipulates that a rotary UAV must be used will rule out this potentially acceptable solution. Even so, we recognize that users will frequently want to request specific equipment packages and we will support that within our Playbook UI.

Next, we will describe how Playbook's APC develops a plan within the specified request constraints provided by the user, as well as general execution constraints such as route, fuel and flight times.

Planning in Playbook

When *Overwatch* is commanded for a fixed target, Playbook's planning component attempts to create a valid, viable path through the template outlined above—a path that corresponds to an executable plan to provide the service as requested by the human user. Let us consider an example in which the user requests *Overwatch* of a specific urban intersection as illustrated in Figure 2 above. The UI sends this request to Playbook's Analysis and Planning Component (APC) as an XML-RPC procedure call.[1] The procedure call contains the information specified by the user and the default parameter values that the user allows to remain in the play template.

In this case, the request specifies *Overwatch* and includes the latest acceptable time on target (say, 0700) and the earliest acceptable time off target (0830). Also in this case, the user has chosen not to specify any constraints on the vehicle or the sensor to be used. This has the effect of leaving the APC more freedom to compose to create plays that meet the user's stated needs, as we shall see below. On the other hand, if the user had made these specifications, Playbook's APC would have attempted to create plans that met them, or would have reported it's inability to do so. In future work, we will strive to include specifications or constraints on the resolution and angle of the imagery to be provided

Playbook's Analysis and Planning Component (APC) is a Hierarchical Task Network (HTN) planning system. The core of the APC is the open-source HTN planner SHOP2. An HTN planner generates mission plans by decomposing goals and tasks into subtasks. In this case, we will see that the APC decomposes the *Overwatch* task into a sequence of primitive operations that can be executed by the VACS ground control station (under Playbook supervision). We have chosen HTN planning, as distinguished from so-called "first principles" planners, because HTNs allow us to compose plans that agree with standard operating procedures familiar to users

[1] XML-RPC is a protocol that permits remote procedure calls encoded as XML and carried over HTTP. It is a relatively lightweight and easy-to-implement protocol for interprocess communication. See http://www.xml-rpc.com/.

The APC contains a library of plan fragments that correspond to the plays that the user can call. This library is somewhat more detailed than the play template depicted in Figure 3 above. For example, the play template in Figure 3 shows that *Overwatch* can be performed by flying one or more Overwatch sorties. The plan library contains specific information about how to decompose a single *Overwatch* play request into one or multiple sorties, as needed.

Among other things, this library includes information about which plan fragments should be tried first and which later—either because they will do a better job of satisfying the request or because they can be investigated or developed more quickly. This preference structure is currently somewhat simple, but can be made more sophisticated and intelligent as the system is developed. In this case, for example, the plan fragment library includes search knowledge that indicates that the single Overwatch Sortie variant should be tried before the Multiple Sortie variant, since this will require fewer resources and coordination demands, thereby generally yielding a better overall plan.

The APC begins to satisfy our example *Overwatch* request by first attempting to build a plan using a single Overwatch Sortie. The APC consults its world model to discover which assets are available, and what constraints exist on their availability. In our current simulations using Playbook, this world model is an internal database defined in initial simulation configuration for each run. In future networked contexts, the APC would get this information from tactical databases, Mobile Ad-hoc Networks (MANETs) or a Common Operating Picture which included UAV information. It is our assumption that such databases will include dynamic information about individual vehicles including type, equipment package, location and a simple *availability window*, indicating the time at which that becomes available and the time by which it must be relinquished by the user and returned to a *destaging point* (or otherwise relinquished from service).

The APC first uses the availability information to see if any UAV could perform a single Overwatch Sortie that satisfies the users' request, specifically a sortie that will put a UAV on target to deliver *Overwatch* imagery by 0700 and keep it there until 0830. In this initial pass, detailed route planning (with associated waypoint development, fuel consumption and travel time estimates) is not performed; preliminary UAV suitability reasoning is based purely on the availability window reported as a part of the vehicle's world state and on rough estimates of travel time. Detailed route planning will be performed later (as described below) as a check on the plans that APC develops. Note that the APC would be able to backtrack and correct itself if these preliminary estimates led it to believe that a UAV was suitable when, in fact, it was not. From its initial world model, the APC learns that there are, let's say, three UAVs available for tasking within its sphere of authority:

1. a fixed-wing Dakota aircraft which is somewhat far away

2. a GTMax rotorcraft[1] which is nearby
3. an Organic Air Vehicle (OAV) which, although very near, must be deployed by special request to its handling unit—a process that is estimated to take at least 45 minutes[2].

The APC attempts to satisfy the single Overwatch Sortie variant of the plan by finding a single vehicle whose availability window will enable it meet the time on target requirements specified in the request. We have also developed logic, for use in future versions of PVACS, that will automatically take into account the appropriateness of various sensor loads, for example, ruling out UAVs without IR sensors for plays called at night. In this case, all three vehicles have acceptable sensors for daylight video, but their availability is problematic. The GTMax is nearby, but its limited endurance capability will not be able to satisfy the full duration of the user's request. While the Dakota has long endurance, it is far away and will not be able to meet the latest time on target requirement. Similarly, the OAV cannot be on target in time to meet the user's request for *Overwatch* imagery. Thus, none of the vehicles are capable of satisfying the parameters of the request by themselves and the APC is unable to develop a single Overwatch Sortie that will meet the customer's needs with the available resources.

Since the APC could not develop a viable plan for satisfying the user's request via the Single Overwatch Sortie method, it next attempts to use the Multiple Overwatch Sortie method. This method finds a UAV that can provide video of the target for a period that covers the latest acceptable time on target, and performs an Overwatch Sortie using that first UAV. Then it determines how much of the user's time period remains uncovered, and constructs a new *Overwatch* request for the remainder of the period. I.e., the Multiple Overwatch Sortie play says "first fly an Overwatch Sortie that gives initial coverage, and then find a new *Overwatch* (which could itself by of a single or multiple sortie variant) that will satisfy the rest of the user's request."

In our example, recall that the GTMax can begin providing *Overwatch* imagery by the user's latest time on target requirement, but cannot stay on target for the entire period, whereas the Dakota and OAV cannot reach the target in time. So the APC will construct an initial Overwatch Sortie for the GTMax, estimating how long it can stay over the target. For the purposes of our example, let's assume that the GTMax can stay over the target only until 0730.

But now the APC will need to satisfy a new, recursive *Overwatch* request, this time to provide coverage over the target from 0730 until at least 0830. Once again, the APC will try to find a single UAV that can perform an Overwatch Sortie to satisfy the entire request. This time, the APC will be successful, since the Dakota can be on target by 0730.

This brings up the general issue of how APC knows to prefer one method of satisfying the service request over other valid ones. This is, in essence, the "common sense" problem that is notoriously difficult for artificially intelligent systems to solve, and it represents a significant area for growth and tuning of our approach in the future. The current version of the APC relies on ordering heuristics to prefer one method to another. As we mentioned above, the APC prefers plans that satisfy *Overwatch* requests with only a single sortie over those that use multiple UAVs. In choosing UAVs for sorties, the APC currently prefers rotorcraft to fixed wing aircraft due to the stability of the image they can provide. When multiple UAVs of the same type are available, the APC prefers vehicles whose availability window (taking into account travel time) more closely matches the needed scanning period so as not to waste available resources.

Given these heuristics, the APC would attempt to satisfy the second Overwatch Sortie instance by first attempting to use the OAV (because, as a rotorcraft, its stable imagery would be preferred to the fixed-wing Dakota's need to continuously circle the target area). In this case, however, the OAV will not be ready for use when needed, thus that plan fragment fails and APC tries again with the Dakota which, this time, succeeds.

The current heuristic approach is not an entirely satisfactory solution to the problem of finding the best plan. Difficulties arise when composing different heuristics. For example, should the APC really prefer a single fixed-wing sortie to two rotorcraft sorties? In future work, we will associate quality measures with different aspects of plans, and allow the APC to try to generate optimal plans. We do not expect that full optimization will be feasible, for reasons of tractability. Instead, we expect to use a regime where the APC will quickly generate a plan that meets the hard constraints of the user's request (a *satisficing* plan), and then opportunistically use available time to generate better plans using variations of branch and bound techniques.

For example, with better representations and reasoning about the capabilities of various sensors and the scan patterns of the various aircraft, it would be possible for the APC to reason about which of the three aircraft described above would do the best job, in the circumstances, of providing imagery of a quality that the user needs and wants—and then to trade that benefit off (or allow the user to do so) against the potential cost of not providing the user with as full a time window of coverage as requested. We note that the quality measures could, in principle, even be tuned for the specific mission or for user preferences in a pre-mission setting. The user could also interact with the APC in its optimization phase to accept or reject plays that the APC feels are improvements. This more sophisticated

[1] The GTMax is a version of the Yamaha RTMax rotary UAV that has been equipped with a specialized control package developed at Georgia Tech.

[2] We do not yet have a good model of OAV flight dynamics in our simulation to enable Playbook reasoning and control—which is our primary reason for making it "sit out" this portion of our walkthrough.

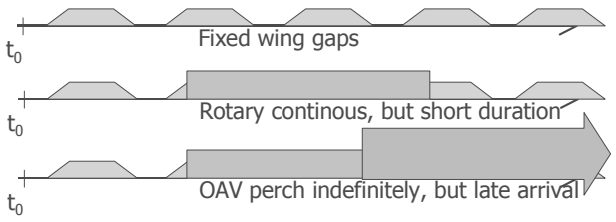

Figure 5. Conceptual relationship between the various coverage options available in our scenario. Coverage is indicated by the horizontal extent of a block; image quality by the height.

APC feels are improvements. This more sophisticated reasoning capability might go on to realize that the OAV, once it becomes available, will provide a better, more stable and more constant image than the Dakota can and, therefore, add a third sortie to the plan to get better coverage later (as illustrated conceptually in Figure 5).

In addition to decomposing the overall *Overwatch* request into two Overwatch Sorties, the APC must decompose the Overwatch Sorties into more primitive action sequences that can be executed by the VACS GCS. Execution will be discussed in more detail in the next section, but examples of executable behaviors include waypoint commands and sensing flight patterns. More sophisticated UAVs could be expected to have more sophisticated execution behaviors and it might be possible for Playbook to create less detailed plans for them.

Consider how the APC plans the GTMax's Overwatch Sortie. First the APC must decompose the Ingress subtask. This part of the planning illustrates an important aspect of the planning process. The first is the fact that the APC integrates constraints from ancillary information sources. In planning an Ingress, the APC will consult an external path planner that conducts an A* optimizing search over a discretized map to find a sequence of waypoints. Note that alternate or special purpose planning algorithms could be incorporated in place of this search method with little problem. The APC incorporates the waypoint specifications into a Fly Waypoint Sequence method. In doing so, it illustrates an additional aspect of the planning process: as the APC builds the sequence of primitive actions (which, in general, includes concurrent actions of multiple UAVs), it reasons about the vehicle's heading, 4D position, velocity, and the passage of time, projecting a *state trajectory* for each vehicle. This is a relatively abstract series of states, linked by discrete transitions, as is appropriate for the APC's role in constructing an outer loop controller for the UAVs.

The plan developed by Playbook serves two purposes. First, it serves as program that will act as an outer loop controller to monitor execution, react to disturbances and replan in accordance with them. Second, portions of the plan will be downloaded to the VACS GCS to preprogram the UAVs' autopilots. The autopilots perform better when programmed with an entire route in advance (permitting later updates), rather than having the Playbook dole out waypoints one at a time. An example sortie plan for one UAV, planned by Playbook and communicated to VACS, is illustrated in VACS Ground Control Station interface in Figure 6. We will discuss Execution monitoring and revision in the next section below.

Execution with Playbook and VACS

The Playbook uses the developed mission plan in two different ways. First, it extracts from the plan a sequence of waypoints that can be downloaded to the Variable Autonomy Control System (VACS) on each UAV. All communications with the vehicles are mediated by the VACS Ground Control Station (GCS). The Playbook Executive component also uses the mission plan as an outer loop control program, enabling it to issue commands to the VACS to perform discrete tasks, such as sensor steering, at appropriate times in the mission. The GCS, in turn, relays vehicle state information from the vehicle control systems to Playbook to enable plan monitoring, closed-loop control, and high-level exception handling and plan repair.

Playbook relies on an execution environment such as VACS to provide intelligent default behaviors which would be assumed by a human tasking a subordinate—such as terrain avoidance, route following, collision avoidance, etc., not to mention such low level control behaviors such as turns, attitude adjustment, etc. Similarly, Playbook does not take responsibility for unexpected conditions that may arise such as loss of communication or some forms of catastrophic vehicle faults. These too are handled by VACS.

Instead, Playbook provides mission-level commanding/requesting capabilities to the human operator—along with the intelligence required to make such commanding at a high level feasible. As described above, Playbook's planning process is carried out to a level of detail sufficient to allow the execution environment (i.e., VACS) to perform the plan with the selected vehicles. In practice, this means that the lowest level of detail that the APC reasons about is within and, typically is the highest of, the levels of control behaviors the execution environment knows how to achieve. Thus, the plans that Playbook passes to VACS are, essentially, a time-sequenced series of executable control behaviors—such as 4D waypoints, scan patterns, etc. Playbook generally does not pass a whole plan to VACS for execution, but rather passes those portions of the plan that are relevant in the current context. This is done, in part, to minimize communication bandwidth.

This interaction between Playbook and VACS also provides the opportunity to do higher level plan monitoring and repairs. VACS, a sophisticated control execution and monitoring environment in its own right, is capable of making many adjustments to "stay on plan". For example, encountering a headwind, VACS will automatically adjust the airspeed of the relevant vehicles to continue to maintain their adherence to 4D waypoints in the plan. But VACS lacks a high-level understanding of the plan as a whole. For example, VACS by itself would be unable to understand that its increase in airspeed will reduce the available

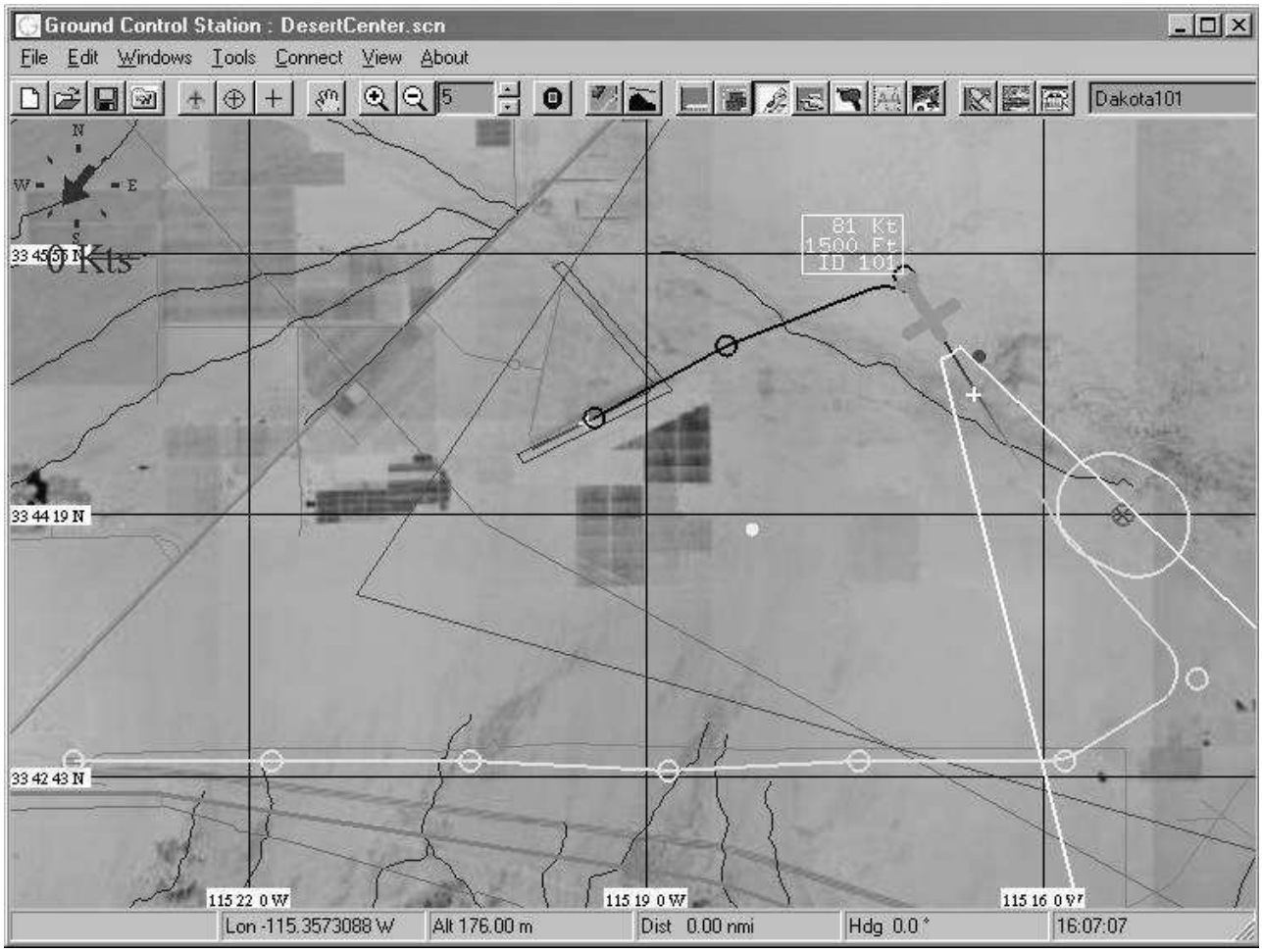

Figure 6. A PVACS-generated and managed route for a Dakota aircraft including ingress from position after takeoff, a racetrack observation position, and egress to a destaging point.

fuel for time on target below the threshold assumed in the original sortie and will, therefore, require the planning of a subsequent sortie to fill in the resulting gap in the user's requested service.

Playbook receives continual updates from VACS about the state of the UAVs involved in Playbook's plans. This allows not only the detection of conditions under which repair plans are needed, but also the triggering of new plan phases. One example is the beginning of the scan phase of the *Overwatch* mission, at which point the UAV's sensor must be steered to provide the desired target coverage. The Playbook will continually monitor the state of the vehicles and compare them with plan-based world state expectations. An Executive component within the APC will use this information to update future expectations and reject disturbances to the plan. The Executive will attempt to maintain the plan on track by making simple, local plan revisions. When these local revisions are inadequate, it will reinvoke the APC to generate a new mission plan. Currently the Executive is relatively minimal, and concentrated on correct execution of the nominal mission plan. Enhancing the control capabilities of the Executive will be a major focus of the next phase of our work. These capabilities will include not only handling disturbances, but also incorporating ongoing user interactions (e.g., requests to tweak sensor targeting).

Conclusions and Future Work

The integration of the mission-level planning and management capabilities of the Playbook approach with the execution environment of the Variable Autonomy Control System shows promise for addressing the critical UAV control needs of the small unit, dismounted soldier engaged in urban combat terrain as identified at the beginning of this paper. Specifically, PVACS provides:

1. the ability for a single operator to manage many UAVs in the satisfaction of his or her requests. In fact, the user need not be concerned with how many or what type of vehicles are used to satisfy requests means that a multiple aircraft plan involves just as much workload on the part of the operator/requester as does a single aircraft plan.
2. the ability for a single, generic workstation to be used to control multiple UAV platforms. Again, the "play calling" approach of PVACS enables a user to request services from a wide variety of

unmanned platforms via the same interface platform—and one that need not be dedicated solely to unmanned vehicle control. We have illustrated how reasonably sophisticated plays can be requested, and refined, for a variety of UAVS via even a device as small and lightweight as a PDA, though obviously a larger display with higher resolution might afford greater control and visualization opportunities.

3. an opportunity to greatly reduce training requirements for UAV control. Because operators no longer need to know specific information about vehicle operating characteristics, training can be greatly reduced. With PVACS, operators only need to learn the set of the plays or requests they can make—and these are driven by their operational needs and even designed around their operating vocabulary, thereby further reducing training time.
4. the ability to control multiple, heterogeneous UAVs while concurrently engaged in other activities. Requesting services by calling plays greatly reduces the workload associated with planning for and controlling multiple UAVs. It remains to be seen whether our approach would be usable and useful even under combat conditions, but our design makes such a scenario plausible, and will support the fine tuning of plays and UI capabilities that might enable that goal.

We have illustrated the ability to control (at least in the sense of requesting a specific service and tuning the fashion in which that request can be satisfied) the behavior of multiple, heterogeneous UAVs in an urban combat environment. While the *Overwatch* play we used in our extended walkthrough is not the most complex form of coordinated UAV behavior that might be desired, it is a start. The PVACS approach provides promise for enabling many more sophisticated behaviors and "plays" which we will explore in future work.

Acknowledgements

This program is funded by a Phase II Small Business Innovation Research grant from DARPA IXO, administered by the U.S. Army Aviation & Missile Command, contract number DAAH01-03-C-R177. We would like to thank Dr. John Bay for his oversight and advice during the investigation. Geneva Aerospace personnel, especially Billy Pate, deserve thanks for their work on the VACS system and for educating us about it.

References

1. Vicente, K., Cognitive work analysis: Towards safe, productive, and healthy computer-based work, Erlbaum, Mahwah, NJ; 1999.
2. Miller, C., Goldman, R., Funk, H. and Parasuraman, R., "Delegation as a Model for Human-Automation Interaction," Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, Houston, Texas, October 27-29, 2002.
3. Miller, C., Pelican, M. and Goldman, R., "'Tasking' Interfaces for Flexible Interaction with Automation: Keeping the Operator in Control," Proceedings of the Conference on Human Interaction with Complex Systems, Urbana-Champaign, Ill. May, 2002.
4. R. Goldman, K. Haigh, D. Musliner, & M. Pelican, "MACBeth; A Multi-Agent, Constraint-based Planner," Notes of the AAAI Wkshp on Constraints and AI Planning, Austin, TX, 2000, pp. 1-7.
5. Parasuraman, R., Galster, S., and Miller, C., "Human Control of Multiple Robots in the RoboFlag Simulation Environment," Proceedings of the 2003 Meeting of the IEEE Systems, Man and Cybernetics society, Washington, DC, October 5-8; 2003.
6. Duggan, David S., "Demonstration of an Integrated Variable Autonomy UAV Flight Control System", Phase II SBIR Final Report, AFRL-HE-WP-TR-2001-0035, January 2001.
7. Pate, B., "A Rotorcraft Adaptation of Geneva Aerospace's Variable Autonomy Control System,".Proceedings of AHS FORUM 60 The Annual Meeting of the American Helicopter Society, Baltimore, MD, June 7-10; 2004.