Miller, Christopher A., Funk, Harry B., Dorneich, Michael C., Whitlow, Stephen, (2002). "A Playbook Interface for Mixed Initiative Control of Multiple Unmanned Vehicle Teams," *Proceedings of the 21st Annual Meeting of the Digital Avionics Systems Conference*, Irvine CA, 27-31 October 2002.

A PLAYBOOK INTERFACE FOR MIXED INITIATIVE CONTROL OF MULTIPLE UNMANNED VEHICLE TEAMS

Christopher A. Miller, Harry B. Funk, Smart Information Flow Technologies, Minneapolis, MN.

Michael Dorneich and Stephen D. Whitlow, Honeywell Laboratories, Minneapolis, MN.

Introduction

Human interaction with automation is becoming increasingly complex as automation becomes more sophisticated. At the forefront of this challenge are settings where one or a few humans, who may be concurrently involved in other tasks such as flying an aircraft, are expected to control, or at least supervise, multi-agent teams of unmanned vehicles. Such interactions place an extreme burden on the information processing and decision-making capabilities of the human(s) involved, and therefore, demand sophisticated methods of communication of intent and of feedback on system performance against objectives, as well as, perhaps, management of the humans' limited resources against the various roles that they and the automated assets can occupy.

There is, however, an existing human activity which roughly matches the challenges and opportunities of this domain and which, therefore, provides a guide for designing human interaction: the role of a quarterback or captain in team sports. Team captains do not command all the actions of all players on their teams, but they do continue to "control" them at a higher level of abstraction by commanding pre-defined plays—plays which define a goal and an acceptable set of behaviors for each player.

In ongoing research on the DARPA Mixed Initiative Control of Automata teams program, we are refining the concept of a 'Playbook Interface' to allow a human to express his or her intent to multiple unmanned vehicles and sophisticated planning and control software to stipulate or constrain the methods that the automated agents use to achieve that intent. Our Playbook Interface is based on a model of the tasks and goals possible in the domain, shared by both humans and automated agents, that facilitates communication about intents and outcomes and can serve as the basis of a wide variety of user interfaces ranging from extremely detailed (for use in pre-mission planning and play development) to extremely streamlined (for in-flight 'play calling'). We will present the architecture required for a Playbook Interface, the various interaction styles it supports, and a preliminary example implementing this approach in an Unmanned Combat Air Vehicle (UCAV) domain.

Problems in Current UCAV Control

Current technologies and approaches require intensive human involvement in the control of unmanned vehicles.[1] Currently deployed Unmanned Air and Combat Air Vehicles (UAVs and UCAVs) in use in Afghanistan are said to require four to five human controllers while in flight, each of whom is fully engaged in various forms of low-level, frequently joystick-based, control of the aircraft, its sensors or other ship's systems. Furthermore, current approaches to the use of UAVs and UCAVs generally require that they fly alone—not only are cooperative teams of vehicles a yet-to-be-achieved vision, but current domestic operations even require that the airspace through which UAVs travel be cleared of other, human-piloted aircraft [1]. While there are various reasons for these restrictions, at least one of them is the reduced complexity associated with a simplified operational environment.

Yet in both military and commercial applications, many dreams of the potential for unmanned vehicles rely on the ability to reduce or even reverse this operational equation. Cooperative teams

[1] While our current work is focused on Unmanned Combat Air Vehicles, much of the argument for the nature of control of multiple vehicles and much of our particular approach applies equally well to any form of unmanned vehicle—air, ground, sea, space, etc.

of unmanned surveillance vehicles hold the potential to enable long term monitoring of a large area for potentially hostile entities and time-critical mobile targets. Large teams of small, portable robots could revolutionize urban warfare, disaster relief, firefighting and hostage situations by providing rapid reconnaissance of a building or other site. Light weight, low cost, high endurance UAVs capable of simultaneously monitoring all of the Amazon or the Indian Ocean are critically needed for the atmospheric and environmental studies needed to address global warming. Finally, the cooperation of a human pilot with one or more UCAV 'wingmen' would provide a suitable mix of human responsibility and oversight, yet limited threat for loss of life, in many near-term military roles.

Yet all of these visions require a substantial change in the way humans interact with unmanned vehicles. Simultaneous and continuous joystick control of multiple vehicles is simply beyond the workload and attentional capacity of human operators. While allowing cooperative teams of unmanned vehicles to be operated on a multiple human to one vehicle basis might be feasible, it would certainly reduce the cost effectiveness and operational roles for such vehicles.

Instead, we must seek ways in which single humans, perhaps already engaged in workload intensive operations of their own (such as flying their own aircraft) can control teams of unmanned vehicles. This is, in fact, the goal of DARPA's Mixed-Initiative Control of Automata teams (MICA) program. MICA seeks to demonstrate human-in-the-loop control technologies that will allow a single human to control five or more UCAVs in an operationally realistic military simulation.

An Informed Delegation Approach

While human control of multiple UCAVs represents a significant, novel challenge for human-automation interaction, the notion of a heavily occupied human "controlling" the action of multiple agents is common in human-human interaction. Human supervisors have long relied on other humans to act for them when out of communication range or under communication restrictions—for example, until very recently in remote business, commercial and legal activities and, still in many cases, in military domains. Similarly, human managers continue to rely on human subordinates to provide the skills and workload capacity to manage large organizations and physical plants of all sorts ranging from aircraft carriers to nuclear power facilities to shopping malls and factory floors.

Perhaps the most obvious and formalized example of such interactions is in sports—where a coach may supervise the actions of a team of many players from the sidelines or a single player (a quarterback or captain) may give "orders" to other team members while him- or herself fully engaged in ongoing activity. Such interaction is made possible by the pre-definition of bounded sets of procedural- and goal-directed activities called "plays". The fact that all team-members share the same definition of a play, combined with the fact that team members can be relied upon to intelligently apply that procedure to the current situation, means that very complex behaviors can be activated with very little time or workload commitment on the part of a human 'supervisor.' We will have more to say about the use of "playbooks" in vehicle control below.

Human-human task delegation has been studied under the headings of communication of intent [2,3] and team situation awareness [4]. The key to good human-human performance in these domains is *informed delegation*—that is, a supervisor's providing of tasking instructions to a subordinate in a way that maintains several attributes:

1. The subordinate has substantial knowledge about and capabilities within the domain. The greater these are the greater the potential for the supervisor to offload tasks (including higher level decision making and course of action selection tasks) on the subordinate.

2. The supervisor is aware of the subordinate's capabilities and limitations and will either not task the subordinate beyond his/her abilities or will provide more explicit instructions and oversight when there is doubt about those abilities.

3. The "language" available for delegation instructions is:

1. easy to use,

2. adaptable to a variety of time and situational constraints,
3. affords discussing tasks, goals and constraints (as well as world and equipment states) as first order objects, and
4. most importantly, is shared by both superior and subordinate.

4. The act of delegation defines a space of control authority within which the subordinate may act. This authority need not be complete (e.g., required checking with the supervisor before proceeding with specific actions or resources), but the greater the authority, the greater the workload reduction on the supervisor.

5. Items 2 and 4 together imply that the space of control authority delegated to automation is flexible—that the supervisor can choose to delegate more or less "space", and more or less authority within that space (that is, range of control options), to automation. Item 3 implies that the language available for delegation must make the task of delegating feasible and robust—enabling, for example, the provision of detailed instructions on how the supervisor wants a task to be performed or a simple statement of the desired goal outcome. Conversely, it is also important that both parties understand the language similarly so that, even when communication is terse, a shared understanding of the delegated control space results.

We have been developing an approach to human-automation interaction that retains the benefits of both automation and of good human-human delegation. This approach is based on the metaphor of a sports team playbook, but the playbook is composed of a hierarchical task model shared between a human user and a variety of planning and control software components. This provides the opportunity for the human to 'task' the automation very flexibly—in all of the ways that s/he might delegate tasks to a knowledgeable human assistant.

Delegation-based approaches provide a variety of payoffs that traditional, static function allocation approaches lack [5]. These include improved situation awareness, more accurate usage decisions, balanced mental workload, increased user acceptance, improved overall human + machine performance and even improved user physical and mental health. Most of these benefits accrue precisely because the human operator can remain actively engaged in the creation, review and monitoring of the activities that even a large team of autonomous agents may perform. The human can be *in charge* of what the team of vehicles does even without being in direct and complete *control* of every action—in the sense that s/he would be using, say, joystick control.

Below, we discuss the architecture and representation required to create a delegation system based on a playbook metaphor. Then we present a usage scenario from an implemented delegation-system prototype illustrating how a "playbook" of shared tasks can allow a human supervisor to "task" a team of UCAVs at a variety of levels.

A Playbook Architecture for Delegation Interactions

Delegation interactions require a shared vocabulary in which task performance can be discussed by human and automation. They further require automation with substantial, autonomous reasoning capability about how to perform tasks and achieve goals within a domain. This same reasoning can be used to improve the safety and efficacy of plans developed by allowing the automation to review and critique human plans. Finally, a "playbook" approach to delegation streamlines the delegation interaction by offering a compiled set of plans, or 'plays', with short, easily-commanded labels that can be further modified as needed.

There are three primary challenges involved in

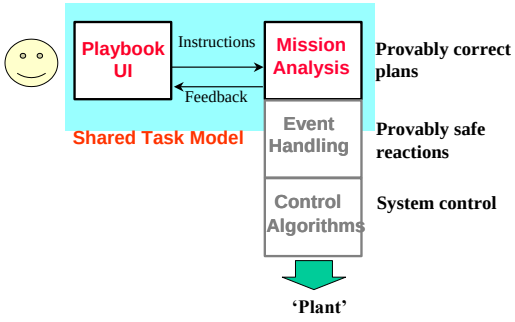

Figure 1. General delegation system architecture.

constructing delegation system:

1. A shared vocabulary must be developed, via which the operator can flexibly pose tasks to the automation and automation can report how it will perform them.
2. Sufficient knowledge must be built into the automation to enable it to make intelligent choices within the instructions provided.
3. One or more user interfaces to permit inspection and manipulation of the vocabulary to pose and review tasks rapidly and easily.

Figure 1 presents our general architecture for delegation systems. The three primary components each address one of the challenges described above. A User Interface (UI) in the form of a "Playbook" and a Mission Analysis Component (MAC) communicate with each other and with the operator via a Shared Task Model. The operator delegates by posing instructions in the form of desired goals, tasks, partial plans or constraints, via the Playbook UI, using the task structures of the shared task model. The MAC is an automated planning system which understands these instructions and (a) evaluates them for feasibility and/or b) expands them to produce fully executable plans. The MAC may draw on special purpose tools (such as a route planner) to perform these functions, wrapping them in its task-sensitive environment.

Outside of the delegation system, but essential to its use in controlling or managing unmanned vehicles, are two additional components. Once an acceptable plan is created, it is passed to an Event Handling component, which is a reactive planning system capable of making moment by moment adjustments to the plan during execution. The Event Handling component then passes these instructions to traditional control algorithms that actually effect behaviors via controlled system automation (sensors and effectors) in the traditional manner.

These components are described in more detail in other publications [5, 6, 7, 8]. Here, we will focus on the core representation and concepts that enable the expression of delegation instructions—the calling of a 'play' in our playbook system.

Shared Task Model

A critical technology for delegation systems is the ability to represent and reference the goals and plans users have in operating automation. By explicitly representing these entities in a format that is familiar yet interpretable by a planning system, we gain a level of human/system coordination beyond that previously possible. We call such a representation a *task model*, because it models the tasks or methods which are known means of accomplishing desired ends within a domain. Figure 2 is a graphical depiction of a small portion of a hypothetical task model for UCAV operations.

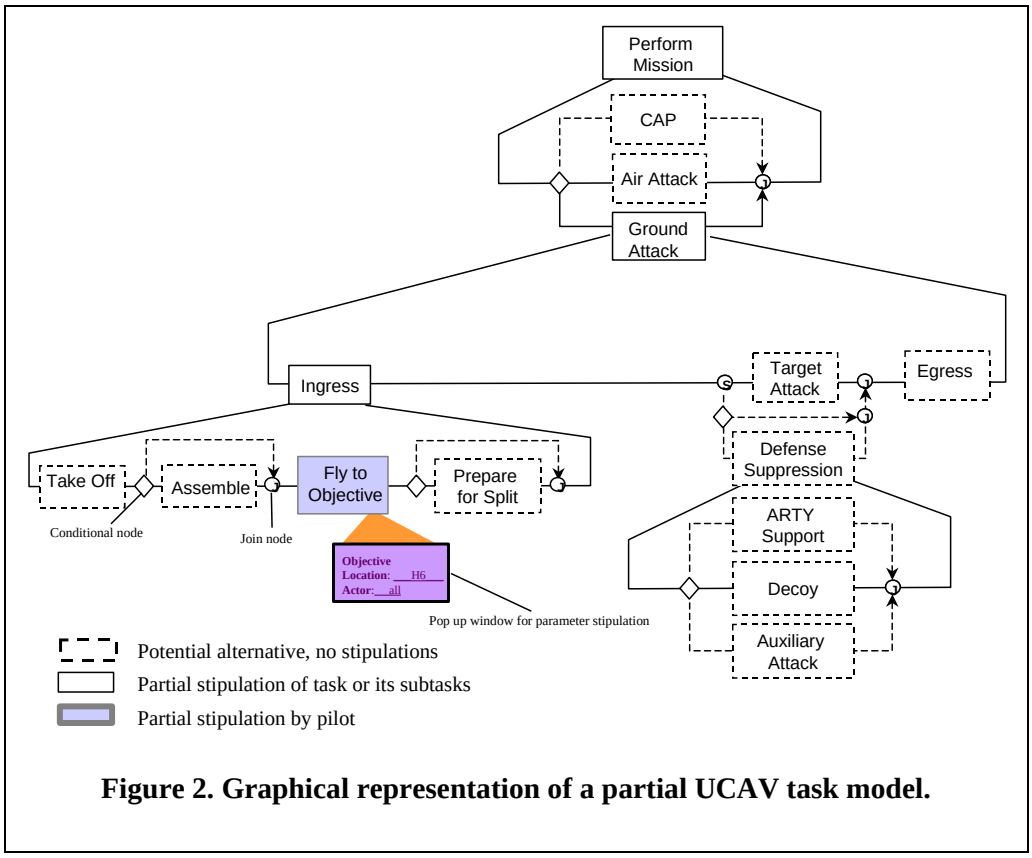

Figure 2. Graphical representation of a partial UCAV task model.

A Task Model Formalism

A task model must meet several requirements to support a delegation system. First, it must represent some set of tasks that the *system* (in this case, a team of UCAVs) is capable of performing. This is unlike traditional human factors approaches to task analysis [9] where the human's actions are the focus of the analysis and the resulting model. The reason for encoding system tasks in the task model is because the model will serve as the framework for delegation interactions. The human will use this task model to command system tasks, hence it must contain and be focused on those tasks rather than tasks the human is performing.

Task models are hierarchical and contain partially ordered sequential constraints. Task models should also, generally, include conditional branching logic. The model in Figure 2 illustrates all of these properties. At its highest level is a single, parent task "Perform Mission". Perform Mission, however, can be decomposed into various methods or types of mission performance—in this simple model, limited to Combat Air Patrol (CAP), Air and Ground Attack missions. The diamond which splits the flow lines into three alternate paths through each of these mission types indicates that these high level tasks are related in an "OR" fashion—generally, only one can be done at a time. Furthermore, the fact that there is no flow path that does not include at least one of these paths indicates that (if this were our complete model) at least one of these high level mission tasks would have to be assigned in any mission the UCAVs were to fly.

The Ground Attack mission task is expanded further in Figure 2. The expansion indicates that Ground Attack must consist of Ingress, Target Attack and Egress sub-tasks and that it may also include an optional Defense Suppression sub-task. The expansion of the Ingress subtask illustrates sequential and conditional task relationships: there must be a Take-Off subtask and it must precede all other subtasks. On the other hand, the conditional split associated with an empty branch around Assemble indicates that this task may either be performed or not, but it's location indicates that, if performed, it must come before Fly to Objective.

By contrast, the expansion of the Defense Suppression task illustrates functional decomposition. The conditional branch here indicates that, if Defense Suppression is done, there are three known methods of performing it: the ARTY Support, Decoy and Auxiliary Attack sub-tasks. Any one of these can be, but one of them must be, used if the Defense Suppression task is to be accomplished.

Primitive Tasks and Stopping Criteria

A task model used for delegation is intended to provide both humans and automation a shared language for talking about tasks to be performed in the domain. It is also intended to encode the knowledge that the automation needs in order to reason about the domain in the same way that the human does. In some sense, the intent of the task model is to give planning or control automation the capability to participate in a pre-mission briefing in the same fashion that a human pilot can. When a commander calls a team together and says "Today we're going to fly a Ground Attack mission," s/he can be reasonably certain that the pilots all share a common understanding of what that means, of what kinds of activities they are likely and unlikely to be doing during the mission, of the range of parameters remaining to be specified before they'll really be able to fly the mission, and of what parameters they'll be called upon to specify and decisions they'll make themselves during flight.

The decomposition employed by the task model must provide these things as well. As can be seen in Figure 2, when flight or planning automation understands this task model, it will know (as a human pilot would) that a "Ground Attack" mission will consist of Ingress, Attack and Egress sub-tasks. It also knows that it must be told what the target of the attack is, but that it may decide what route to take to get there, etc.

The question of how deeply to decompose tasks is inevitable in task modeling [10]. The finest level tasks in a decomposition are frequently called *primitive tasks*, and the conceptual level at which decomposition ceases is determined by *stopping criteria*. Stopping criteria may be practical as much as theoretical—if there is no need to make a finer distinction or maintain a model of tasks below a given level, then that level is a fine one to stop at.

For delegation systems, we actually have two stopping levels that, though they may sometimes be synonymous, are conceptually distinct. *Human Primitive Tasks* (HPTs) are the lowest level at which a human operator can or would want to inte-

ract with the delegation system. *Automation Primitive Tasks* (APTs) are the lowest level at which the automation needs to reason about performance to effect behavior. Typically, APTs must be executable by existing control software. Alternatively, as in our Playbook, if Event Handling software exists to reason in real time about low-level operator selection, the then APTs will need to reach the input level for the Event Handling software.

For example, for a given application the human operator may want or need to dictate low-level motion commands such as Achieve Speed X (with a specific speed parameter value) or Maintain Attitude Y. These would then be HPTs for that application. For another application, these low level tasks may be unnecessary and the HPTs might, instead, be at the level of Move to Position. In either case, in order to actually effect movement behavior, the automation would need to reason beyond even speed and attitude settings to still lower level tasks such as Set Flaps, Set Trim Tabs, Adjust Fuel Injectors, etc. These are the sorts of tasks that will, typically, be at the APT level. Generally, HPTs will exist higher in the hierarchy than APTs but the two levels may be identical for some applications.

Task Model States

In order to support delegation reasoning, the task model used by a delegation system must be capable of existing in several different states. Before any mission planning or human delegation has been done, the task model exists in a completely uninstantiated state. We call this an *General Task Model* (GTM). This corresponds loosely to the knowledge a pilot might have before coming into the briefing room. S/he knows a great deal about what constitutes a Ground Attack task—which subtasks it can entail and in what orders, what subtasks simply will not be a part of it (because, if they were used, the mission would be called something else), which equipment is likely to be used and which would be nonsensical, even how long it is likely to take—but s/he knows nothing about this particular mission including whether or not it will contain a Ground Attack task.

As the mission commander and the pilots begin to discuss today's mission, they create a specific instance of the GTM. The top-level node for a mission will always be the Perform Mission task shown in Figure 2, but the lower level tasks intended for this particular mission will differ. Hence, delegation in the context of mission planning means developing a shared, specific instance of the GTM where specific tasks are highlighted as being intended for performance. We call this instance a *Specified Task Instance Model* (STIM).

The process of specifying the GTM to produce a STIM involves two different types of actions: task selection and parameter specification. Wherever choices of tasks occur in the GTM, specific options must be chosen. For example, in Figure 2 above, a choice must be made as to which high level task will be the focus of the mission: CAP, Air Attack or Ground Attack. Similarly, under Ingress, a choice must be made as to whether or not this STIM will include an assemble task or not.

The second type of specifying action is to fill in parameter values for the tasks which are chosen. An uninstantiated task in the GTM is called a *task template*. Task templates, in fact, define a range of behaviors that we have agreed to label with the name of the task. For example, the Achieve Speed task described above is a general task template suitable for commanding any speed of which the aircraft is capable. Instantiating the generic template involves inputting a specific value (or range) for this instance of the task. A particular set of parameter inputs is illustrated in Figure 2 for the Fly to Objective task— the objective's location. Particularly problematic parameter values generally included for all tasks in the model are a start and end time (or duration). While these are clearly important things to know about a task, they are very difficult to stipulate a priori for many tasks, especially low level ones.

The process of instantiating the GTM to make it an STIM proceeds both in time (sequentially through the time planned for the mission events) and in depth along the decomposition dimension of the task model. A *Fully-Specified Task Instance Model* (Full STIM) is one in which all tasks have been decomposed and specified to the APT level. All task options have been selected and all specific parameter values have been chosen. The only practical examples of Full STIMs are mission traces—that is, histories of missions actually flown. While it would be possible to create an Full STIM for a mission, it would be nearly useless as a planning or delegation artifact because it would have been

over-planned. It would include tasks like Achieve Flap Setting with parameter values like Setting = 3.17 degrees at Start Time = 13:42:27.16. To plan to this level of detail much in advance is clearly useless because the need for this flap setting can only be determined fractions of a second before it is needed. While it would be possible for a mission commander to make such a plan, it would be a waste of time because it would be invalidated long before the pilots left the ground.

Thus, in delegation, we almost always work with *Partially Specified Task Instance Models* (Partial STIMs). These are STIMs in which only some of the choices have been made, leaving the rest as free variables to be decided later. A Partial STIM is not a full plan in the artificial intelligence sense, but it may well be a full mission plan in the sense that a commander might give it to a pilot—leaving many decisions to be made during flight. Any mission which is only decomposed to the HPT level will necessarily be a Partial STIM, but delegation can easily be done with Partial STIMs that are not decomposed all the way to HPTs. From the figure above, the commander could stipulate that this mission (that is, the Partial STIM under development) will involve a Ground Attack task but not provide any further stipulation about whether or not the mission will include a Defense Suppression task, leaving that decision to his/her pilots.

A special type of Partial STIM is an *Executable STIM*. This is a Partial STIM that has been defined, by human and/or machine, to a complete enough degree to make it executable by the control system. Here, enough of the possible variation in what could constitute a mission has been declared for the system to be able to do the rest. Just as the commander would not expect to walk into the briefing room and tell his/her pilots only "okay, today I want you to fly a mission. Go to it," so a delegation system needs at least a bit more framing information.

This is the goal of pre-mission planning—to get the intended STIM to a point where the automated planning and/or control software can execute a mission which fulfills it. What is needed in order to make a Partial STIM an Executable STIM is, as might be expected, a function of the planning and control software. If, say, a set of waypoints is required before a path planner can create a path for a UCAV, then the Partial STIM must stipulate tasks which include that information. Sophisticated planning software, such as we have been developing for the interface described below, may well be capable of creating a executable plan even from very high level tasks in the hierarchy. In practice, however, we frequently require certain parameters to be stipulated by the human even though planning software might be capable of deciding them on its own. Target designation is one such example.

Play Calling

The shared vocabulary of tasks, their instantiation parameters and the relationships between them, therefore, provides a means of communication between user and system, permitting delegation. Better yet, if (as in our playbook system), the vocabulary can be used by a planning system to construct valid strings of sub-tasks to accomplish a parent task, then the architecture supports a highly flexible form of delegation interaction very similar to the relationship a mission commander can have with the well-trained pilots in his/her squadron.

We refer to the delegation interaction between a human operator and automated software (whether onboard, offboard or a combination thereof) controlling one or more UCAVs as *play calling*. The operator who 'calls plays' must interact directly with the task model, activating and combining tasks at various levels of decomposition. This capability is provided via the Playbook UI, though the nature of that UI and the levels and combinations of plays available may differ from application to application. We also provide a planning system, the Mission Analysis Component depicted in Figure 1, that can understand the operator's tasking commands and either evaluate them for performability or, develop an executable plan that obeys, yet fleshes them out.

The operator must interact with the task model, both to understand possible actions and, more importantly, to declare those tasks, goals, partial plans and constraints s/he wishes the system to pursue. Just as a quarterback or team captain can activate a complex behavior by referring to a simple play name or can spend additional time combining play elements or tweaking parameters, so operators are able to tune their interaction with automation via the Playbook UI to fit available time and contexts.

In practice, this means that the operator begins with a generic and uninstantiated version of the task model—the GTM described above. The Playbook UI must enable the two actions described above for turning a GTM into a Partial STIM:

1. Play selection from among viable alternatives, and
2. Play parameter instantiation.

The set of selection and parameterization actions performed must result in an Executable STIM if the resulting plan is to be flown.

The Role of the MAC

In our approach, these tasks are not performed by the human commander alone. Instead, the human interacts with the Mission Analysis Component (MAC) illustrated in Figure 1. The MAC is a planning system which understands and uses the task model to create feasible plans to the level of input required by the Event Handling software. Since it 'speaks the same language' of tasks as the human, it is capable of taking directions from the user and planning within them.

The MAC [6] operates over Partial STIMs provided by the operator to: (1) analyze the operator's plan for feasibility and goal achievement and (2) automatically generate candidate plan completions in keeping with the partial plan the human imposes. The MAC can critique [11] the operator-specified plan for feasibility and constraint violations and weed out candidate sub-plays that have been made infeasible by earlier decisions. Finally, the MAC can complete a partial plan (that is, produce an Executable STIM) from whatever level the human chooses to hand over. The MAC will either incorporate and obey those portions of the plan the human specified, or report why an executable plan cannot be completed within constraints.

The MAC uses a hierarchical task network planner [12] in conjunction with constraint propagation techniques [13, 14] to perform the functions described above. By using the same structures of the task model that the human uses, the MAC's concepts of available 'plays' necessarily mirror those of the user. In turn, the MAC must manage the resources, deadlines, etc., checking for feasibility and conflicts between alternative plan in-stances. The MAC represents these limited quantities as constraints on and between individual plan operators that are maintained by a constraint management engine, but these plan operators are sequenced and composed in conjunction with a human operator.

As the human constructs a plan, the MAC continuously determines the feasibility of the plan. 'Feasibility' is the projected plan's ability to achieve the declared goal state (the top level task) within resource limitations. When asked to check for feasibility or complete a plan, the MAC fleshes out the non-APT tasks in the plan by asserting one or more subtask methods that can fulfill the parent goal. When critiquing, the MAC provides feedback on the feasibility of the currently specified plan. The MAC can aid decisions by having feasible plays at the next decomposition level be presented and infeasible ones eliminated or 'grayed out,' if desired. Alternatively, when in plan completion mode, the MAC can select its best completion according to resource usage criteria. The UI then displays the planning decisions to the user, who can retract choices, or make better-informed decisions from among the available, feasible plans.

In combination, feasibility checking and plan expansion make it possible, but not required, for the MAC to generate effective plans with a minimum of user involvement. Continual feasibility analysis minimizes the effort expended on dead-ends while encouraging the user to specify the mission critical details as early as possible. Once these are stipulated, the development of the plan can be left entirely to the MAC with the assurance that it will produce a plan that is both feasible within its constraint knowledge and in keeping with the operator's stipulations. If time permits (or lack of trust demands), the user may provide increasingly detailed instructions by selecting among available plan alternatives, down to the HPT level.

The Playbook UI

The user's interaction with the task model and the MAC is via a user interface (UI). Some requirements for the Playbook UI include: (1) the set of tasks (e.g., maneuvers, procedures, etc.) represented must be those any well-trained operator should know, (2) the general task templates can be composed and instantiated to create many specific mission plans, (3) the operator may select tasks at various hierarchical levels, leaving the lower levels to be composed by the MAC, and (4) operators may either require or prohibit the use of specific tasks or of specific resources for a task.

One of the strengths of using the shared task model as an infrastructure is that it enables a wide diversity of UIs—each customized for their context of use. Figure 3 illustrates some potential usage alternatives and describes the level(s) of the Task Network they would likely interact with.

For example, a commander responsible for a large number of assets (typically, one of higher rank—say, Lt. Col. and above) might be constrained to delegate only at the higher levels of the network and to leave those assets more autonomy to develop their plans at the lower, executable layers of the network. By contrast, a lower level commander (say, captain or below), would likely want to very carefully task assets and review even the lowest levels of the plans they create.

Both of those examples presume that delegation and tasking is done a priori, during a mission planning phase; but the task network also supports very dynamic, in flight delegation through play calling. Here, plays are labels associated with intermediate-level tasks in the network. The label references a constrained range of variance for the tasks beneath it. By 'calling the play' (that is, activating or referencing the label), the human authorizes the automation to perform any variation of the sub-tasks which fall under that heading. For example, a human pilot in flight might command an unmanned wingman to "Reconnoiter" with a specific area as a required parameter value. While this is a very speedy delegation command for, potentially, a very complex behavior, it would leave the UCAV with the authority to take any path or speed it deemed appropriate to perform that action. The tradeoff would involve improved speed of delegation with decreased sensitivity in what can be commanded. Again, similar to play calling on a basketball court, vs. play calling during a time out on the side lines vs. play development back in the locker room between games (an activity that may require special tools from a UI).

Some generally useful attributes of a delegation system's UI are described below. In the following section, we describe one Playbook prototype we have developed for ground-based planning of UCAV missions.

First, a delegation system UI must include some ability to access and command pre-defined tasks from a library, usually at various hierarchical levels. Second, most applications will benefit from more elaborate (and sensitive) communication than simply accessing a pre-defined task. This can be provided minimally by allowing the operator to instantiate the parameters of a task. More elaborately, plays/tasks may need to be composed into longer sequences (e.g., missions). A mission plan composition workspace and tool separate from a 'play calling' tool will help in these cases. Third, many domains will require creating new tasks or plays, either from scratch or by storing the results of earlier composition. A different tool, or mode, should support this type of interaction. Fourth, most domains will need to visualize the performance and outcome of commanded plays. Normal automation interfaces may provide these in a raw form, but referencing performance against the intended plays should improve user understanding. Finally, the UI must support interaction with the MAC via issuing partial tasking instructions for completion, receiving critiques, and previewing and accepting or modifying MAC-generated plans.

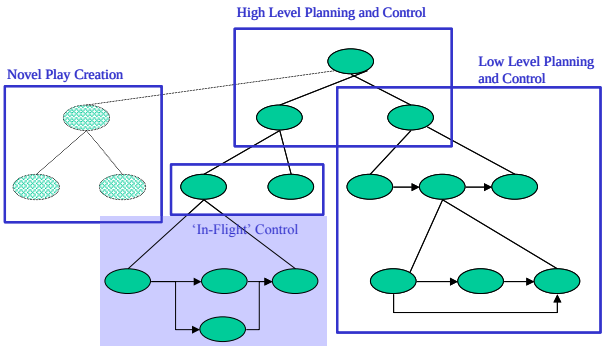

Figure 3. Different uses of the task model.

Interacting directly with an explicit task model (as illustrated below) meets most of these requirements, but we have found that it helps to make the UI multi-modal. Visualization of the task model shows causal and sequential relationships, it does not do a good job of conveying the particular assets involved in each task, temporal duration of events, geographical location and progression of events and objects, etc. Furthermore, as Oviatt [15] has found, interaction with a domain-specific visualization such as a map in the context of a known task can be a very efficient method of specifying and visualizing task parameters.

Usage Scenario

The following scenario illustrates how a user might interact with the delegation prototype we have developed to plan a UCAV mission. Building on prior Honeywell control algorithms and simulation work supporting scenarios of multiple uninhabited F-16s, we developed a delegation interface to enable a human leader to lay out a mission plan. This interface will support the stipulation of full and partial plans and constraints for the UCAVs either separately or in conjunction. To date, we have concentrated on a ground-based tasking interface due to its lighter demands on user, simulation and interface design. However, we believe that suitable interface modifications will suit this approach to in-flight tasking as well.

Figure 4 shows the five primary regions of the prototype Playbook UI. The upper half of the screen is a Mission Composition Space that shows the Partial STIM composed thus far. The lower left corner of the interface is an Available Resource Space, currently presenting the set of aircraft available for use. The lower right corner contains an interactive Terrain Map of the area of interest, used to facilitate interactions with significant geographic information content. The space between these two lower windows (empty at startup) is a Resource in Use Space—once resources (e.g., UCAVs, munitions, etc.) are selected for use, they will be moved to this workspace, where they can be interacted with in more detail. Finally, the lower set of control buttons is always present for interaction with the system. This includes options such as "Finish Plan" for handing the partial plan off to the MAC for completion and/or review and "Show Schedule" for obtaining a Gantt chart timeline of the activities planned for each actor, etc.

At startup, the Mission Composition Space presents the three top-level plays (or 'mission types') the system currently knows about: currently, Interdiction, Airfield Denial, and Suppress Enemy Air Defenses (SEAD). The mission leader would interact with the playbook to, first, declare that the overall mission task for the day was, say, "Airfield Denial." In principle, the user could define a new top-level play either by reference to existing task structures or completely from scratch, but this capability has not been implemented yet.

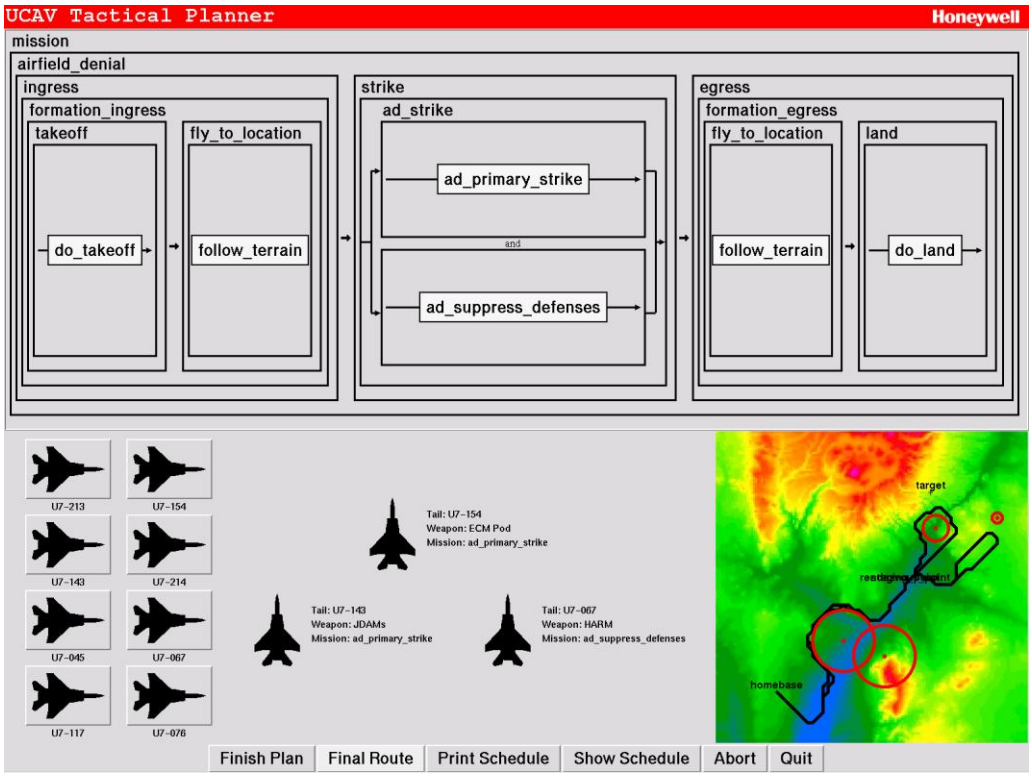

Figure 4. Prototype Playbook Interface for UCAV Mission Planning.

Clicking on "Airfield Denial" produces a pop-up menu with options for the user to tell the MAC to "Plan this Task" (that is, develop a plan to accomplish it) or indicate that the user will "Choose airfield denial" as a task that s/he will flesh out further. The pop-up menu also contains a context-sensitive list of optional subtasks that the operator can choose to include under this task. This list is generated by the MAC with reference to the existing task structures in the task model, filtered for

current feasibility.

At this point, having been told only that the task for the day is "Airfield Denial," a team of trained pilots would have a very good general picture of the mission they would fly. Similarly, the delegation system (via the Shared Task Model) knows that a typical airfield denial plan consists of ingress, attack and egress phases and that it may also contain a suppress air defense task before or in parallel with the attack task. But just as a leader instructing a human flight team could not leave the delegation instructions at a simple 'Let's do an Airfield Denial mission today,' so the operator is required to provide more information. Here, the human must provide four additional items: a target, a homebase, a staging and a rendezvous point. Most of these activities are geographical in nature and users typically find it easier to perform them with reference to a terrain map. Hence, by selecting any of them from the pop up menu, the user enables direct interaction with the Terrain Map to designate an appropriate point. Since the Playbook knows what task and parameter the point is meant to indicate, appropriate semantics are preserved between user and system. As for all plans, the specific aircraft to be used may be selected by the user or left to the MAC. If the user wishes to make the selection, s/he views available aircraft in the Available Resource Space and chooses them by clicking and moving them to the Resources in Use Area.

The mission leader working with a team of human pilots could, if time, mission complexity or degree of trust made it desirable, hand the mission planning task off to the team members at this point. The playbook operator can do this as well, handing the task to the MAC via the "Finish Plan" button. The leader might wish, however, to provide substantially more detailed delegation instructions. S/he can do this by progressively interacting with the UI to provide deeper layers of task selection, or to impose constraints or stipulations on the resources to be used, way-points to be flown, etc.

For example, after the user chooses 'Airfield Denial' the system knows, via the Shared Task Model, that this task must include an Ingress subtask (as illustrated in Figure 4). To provide detailed instructions about how to perform the Ingress task, the user must choose it, producing a "generic" Ingress task template from the GTM. This is not a default method of doing "Ingress" but a generic, uninstantiated template—corresponding to what a human expert knows about what constitutes an Ingress task and how it can or should be performed. A trained pilot knows that Ingress can be done either in formation or in dispersed mode and, in either case, must involve a "Take Off" subtask followed by one or more "Fly to Location" subtasks. Similarly, the playbook user can select from available options (e.g., formation vs. dispersed Ingress, altitude constraints on takeoff, etc.) on context-sensitive, MAC-generated menus appropriate to each level of decomposition of the task model.

The user can continue to specify and instantiate tasks down to the HPT level. In practice, in order to preserve control stability, it may frequently be the case that the HPT level is not synonymous with the APT level where the sub-tasks are behaviors the control algorithms (see Figure 1) in our simulator can be relied upon to execute in flight. The MAC and the Event Handling component are, collectively, responsible for driving the plan to the APT level and creating an Executable STIM.

In practice, however, users will frequently be willing to stop planning before reaching even the HPT level. This may be because the user trusts the system to be able to develop an acceptable plan, or because the current situation doesn't require a particularly sophisticated or sensitive plan, or because s/he does not have time to develop a plan to the granularity of the HPTs—a case that is of particular interest in trying to enable a single commander to control and task multiple UCAVs. Our approach supports this capability by allowing, at any point after the initial selection of the top level mission task and its required parameters, the tasker to hand the partly developed plan over to the MAC for completion and/or review. In extreme cases, a viable "Airfield Denial" plan could be created in our prototype with as few as five selections and more sophisticated planning capabilities could readily reduce this number further. If the MAC is incapable of developing a viable plan within the constraints imposed, (e.g., if the user has stipulated distant targets that exceed aircraft fuel supplies) it will inform the user of these problems.

Conclusions and Future Work

We are currently at work developing Playbook concepts to support ground-based human mission planning and control of multiple UCAVs for the DARPA MICA project. This work is placing emphasis on more active monitoring and control of UCAVs while they execute a pre-planned mission plan than illustrated in the earlier prototype described herein. Similarly, we are exploring appropriate levels of depth for both APTs and HPTs, and for defining useful collections of tasks which can be called via simple labels as plays.

We are also engaged in research on the use of delegation approaches to in-flight control of UCAVs and of one's own aircraft. This work demands not only consideration of novel user interface concepts but, equally importantly, of control stability issues in delegating tasks at various levels from human to automation or vice versa.

In future work, we are interested in validating improvements provided by delegation systems either in terms of overall system performance, or in terms of human situation awareness, engagement, etc., or both. The literature gives us every reason to believe that such benefits should accrue [5], but we have yet to develop a sufficiently rich human-in-the-loop simulation in order to be able to test them.

Acknowledgements

We would like to thank Dan Bugajski, Don Shaner, John Allen and David Musliner for their help in the development and implementation of the ideas presented. Robert Goldman deserves special mention as instrumental in realizing the operation of the MAC and its integration with a human-understandable task model. Robert Goldman and Michael Pelican were responsible for the specific design and implementation of the example shown in Figure 4. The initial formulation of the Playbook concept and the development of the prototype interface described here was funded by a Honeywell Initiatives Grant. Additional work reported here was funded by the DARPA Mixed Initiative Control of Automata Teams program, under subcontract to Honeywell, and by a DARPA Small Business Innovative Research contract # DAAH01-02-C-R163 to the U.S. Army Aviation and Missile Command titled "Multi-Modal Control for Automatic Vehicle Management Systems" to SIFT.

[1] Clarke, Tom, June 6, 2002, "Flying Free" *Nature*, Vol. 417, pp. 582-583.

[2] Klein, G., 1998, *Sources of Power; How People Make Decisions*, Cambridge, MA; MIT Press.

[3]Shattuck, L.G.and D.D. Woods, 2000, "Communication of Intent in Military Command and Control Systems," In Carol McCann and Ross Pigeau (Eds.), *The Human in Command: Exploring the Modern Military Experience*, New York: Kluwer Academic/Plenum Publishers, pp. 279-292.

[4] McNesese, M., Salas, E. and Endsley, M., 2001, *New Trends in Cooperative Activities.* Human Factors and Ergonomics Society; Santa Monica, CA.

[5] Miller, C. and Parasuraman, R. Submitted. Designing for Flexible Human-Automation Interaction: Playbooks for Supervisory Control. Submitted for publication in *Systems, Man and Cybernetics: Part A—Systems and Humans*.

[6] Goldman, R., K. Haigh, D. Musliner, & M. Pelican, 2000, "MACBeth; A Multi-Agent, Constraint-based Planner," in *Notes of the AAAI Wkshp on Constraints and AI Planning*, Austin, TX, pp. 1-7.

[7] Musliner, D., E. Durfee & K. Shin, 1993, "CIRCA: A cooperative intelligent real-time control architecture," *IEEE Trans. on Sys. Man & Cyber*., vol. 23, pp. 1561-1574.

[8] Miller, C., R. Goldman, & M. Pelican, 2000, "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.

[9] Kirwan, B. and Ainsworth, L., 1992, *A Guide to Task Analysis*, London; Taylor and Francis.

[10] Shepherd, A., 1989, "Analysis and Training in Information Technology Tasks" in Diaper, D. (Ed.) *Task Analysis for Human-Computer Interaction*, Chichester, UK; Ellis Horwood, pp. 15-55.

[11] Guerlain, S., 1995, "Using the critiquing approach to cope with brittle expert systems" *Proc. HFES 39th Annual Mtg*., Santa Monica, CA; October, pp. 233-237

[12] Erol, K, Hendler, J., and Nau, E., 1994, "UMCP: A sound and complete procedure for hierarchical task network planning," in *AI Plan. Sys.: Proc. of 2nd Int. Conf.*, K. Hammond, Ed., Los Altos, CA, pp. 249-254.

[13]Hentenryck, P., 1989, *Constraint Satisfaction in Logic Programming.* Cambridge, MA: MIT Press.

[14] Jaffar, J. and Michaylov, S., 1987, "Methodology and implementation of a CLP system," *Proc. of 4th Int. Conf. Logic Prgmng*, Cambridge, MA: MIT Press.

[15]Oviatt, S., 1998, "User-centered modeling for spoken language and multimodal interfaces," in M. Maybury & W. Wahlster, (Eds.), *Intelligent User Interfaces*, San Fansisco; Morgan-Kaufman, pp. 620-630.