Delegation Architectures:
Playbooks and Policy for Keeping Operators in Charge

Christopher A. Miller

Smart Information Flow Technologies
1272 Raymond Ave
St. Paul, MN 55108 U.S.A.
cmiller@sift.info

Abstract

We argue that as Unmanned Military Vehicles become more intelligent and capable, and as we attempt to control more of them with fewer humans in the loop, we need to move toward a model of delegation of control rather than the direct control (that is, fine grained control with, generally, tight and fast control loops) that characterizes much current practice. We identify and describe five delegation methods that can serve as building blocks from which to compose complex and sensitive delegation systems: delegation through (1) providing *goals*, (2) providing full or partial *plans*, (3) providing *negative constraints*, (4) providing *positive constraints* or *stipulations*, and (5) providing priorities or value statements in the form of a *policy*. We then describe two implemented delegation architectures that illustrate the use of some of these delegation methods: a "playbook" interface for UAV mission planning and a "policy" interface for optimizing the use of battlefield communications resources.

UMV Control as Human-Automation Delegation

While Unmanned Military Vehicles (UMVs—that is, any unmanned vehicle, whether ground, air, sea, undersea or space, used for military purposes) hold the promise of radical change and improvement for a wide range of military applications they also pose a host of challenging problems. Chief among these is how to enable a human operator, who may well be heavily engaged in other tasks of his or her own (such as exploring a building, maintaining radio contact with headquarters or even avoiding fire), to retain sufficient control over the UMV(s) to ensure safe, efficient and productive outcomes. This problem is, of course, magnified when the UMVs may be responsible for the lives of many soldiers or civilians, may be capable of unleashing lethal force on its own, and when a single human may be striving to control groups or even swarms of potentially autonomous and independent actors and may be concurrently engaged in other, high tempo and criticality tasks of his or her own.

Yet this problem is not completely novel. Humans have been striving to retain control and produce efficient outcomes via the behavior of other autonomous agents for millennia. It just so happens that those "agents" have been other humans. Not surprisingly, we have developed many useful methods for accomplishing these goals, each customized to a different domain or context of use. When we have some degree of managerial authority over another human actor and yet will not be directly commanding performance of every aspect of a task, we call the relationship (and the method of commanding task performance) *delegation*. Delegation allows the supervisor to set the agenda either broadly or specifically, but leaves some authority to the subordinate to decide exactly how to achieve the commands supplied by the supervisor. Thus, a delegation relationship between supervisor and subordinate has many requirements:

1. The supervisor retains overall responsibility for the outcome of work undertaken by the supervisor/subordinate team and retains the authority commensurate with that responsibility.
2. The supervisor has the capability to interact very flexibly and at multiple levels with the subordinate. When and if the supervisor wishes to provide detailed instructions, s/he can; when s/he wishes to provide only loose guidelines and leave detailed decision making up to the subordinate, s/he can do that as well—*within the constraints of the capabilities of the subordinate*.
3. To provide useful assistance within the work domain, the subordinate must have 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 tasks) on the subordinate.
4. The supervisor must be aware of the subordinate's capabilities and limitations and must either not task the subordinate beyond his/her abilities or

must provide more explicit instructions and oversight when there is doubt about those abilities.

5. There must be a "language" or representation available for the supervisor to task and instruct the subordinate. This language must (a) be easy to use, (b) be adaptable to a variety of time and situational contexts, (c) afford discussing tasks, goals and constraints (as well as world and equipment states) directly (as first order objects), and (d) most importantly, be shared by both the supervisor and the subordinate(s).
6. The act of delegation will itself define a window of control authority within which the subordinate may act. This authority need not be complete (e.g., checking in with the supervisor before proceeding with specific actions or resources may be required), but the greater the authority, the greater the workload reduction on the supervisor.

Items 4 and 6 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 5 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.

Types of Delegation

We have developed a variety of architectures within which to support human delegation interactions with automation. Of particular interest as a core enabling technology is the "language" or representation for delegation described in item #5 above. As Klein (1996) points out, without successfully sharing an understanding of the tasks, goals and objectives in a work domain, there can be no successful communication of intent between actors. We believe there are five kinds of delegation actions or delegation methods that should be supported within such a representation, as described in Table 1 below. Note that each method forms a building block, and they can be combined into more effective and flexible composite delegation interactions. Note also that the subordinate has a specific responsibility in response to each method, as articulated below.

In the remainder of this paper, I will described two delegation architectures we are developing. While neither system enables all of the types of delegation described above, and neither is fully implemented yet, collectively they illustrate the five types of delegation and provide a rich and highly flexible set of interactions for human-automation delegation.

Playbook—Delegation of Goals, Plans and Constraints

The first architecture is based on the metaphor of a sports team's playbook. A playbook works because it provides for rapid communication about goals and plans between a supervisor (e.g., a coach) and a group of intelligent actors (the players) who are given the authority to determine how to act within the constraints inherent in the coach's play. Our Playbook architecture supports delegation action types 1-4 in principle and has been implemented in prior prototypes to include action types 2 and 4.

The basic Playbook system architecture is presented in Figure 1. The Playbook 'proper' consists of a User Interface (UI) and a constraint propagation planner known as the Mission Analysis Component (MAC) that communicate with each other and with the operator via a

Table 1. Five types of delegation.

Supervisor's Delegation Action	Subordinate's Responsibility
1. Stipulation of a goal to be achieved—where a goal is a desired (partial) state of the world.	Achieve the goal(s) if possible (via any means available), or report if incapable.
2. Stipulation of a plan to be performed—where a plan is a series of actions, perhaps with sequential or world state dependencies.	Follow the plan if possible (regardless of outcome) or report if incapable.
3. Provide constraints in the form of actions or states to be avoided.	Avoid those states or actions if possible, report if not.
4. Provide "stipulations" in the form of actions or states (i.e., sub-goals) to be achieved.	Achieve those states or perform those actions if possible, report if not.
5. Provide an "optimization function" or "policy" that enables the subordinate to make informed decisions about the desirability of various states and actions	Work to optimize value within the "optimization function" or "policy".

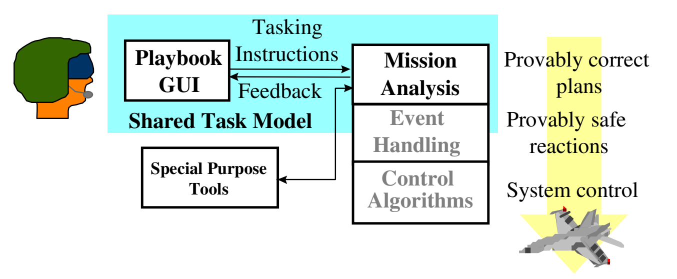
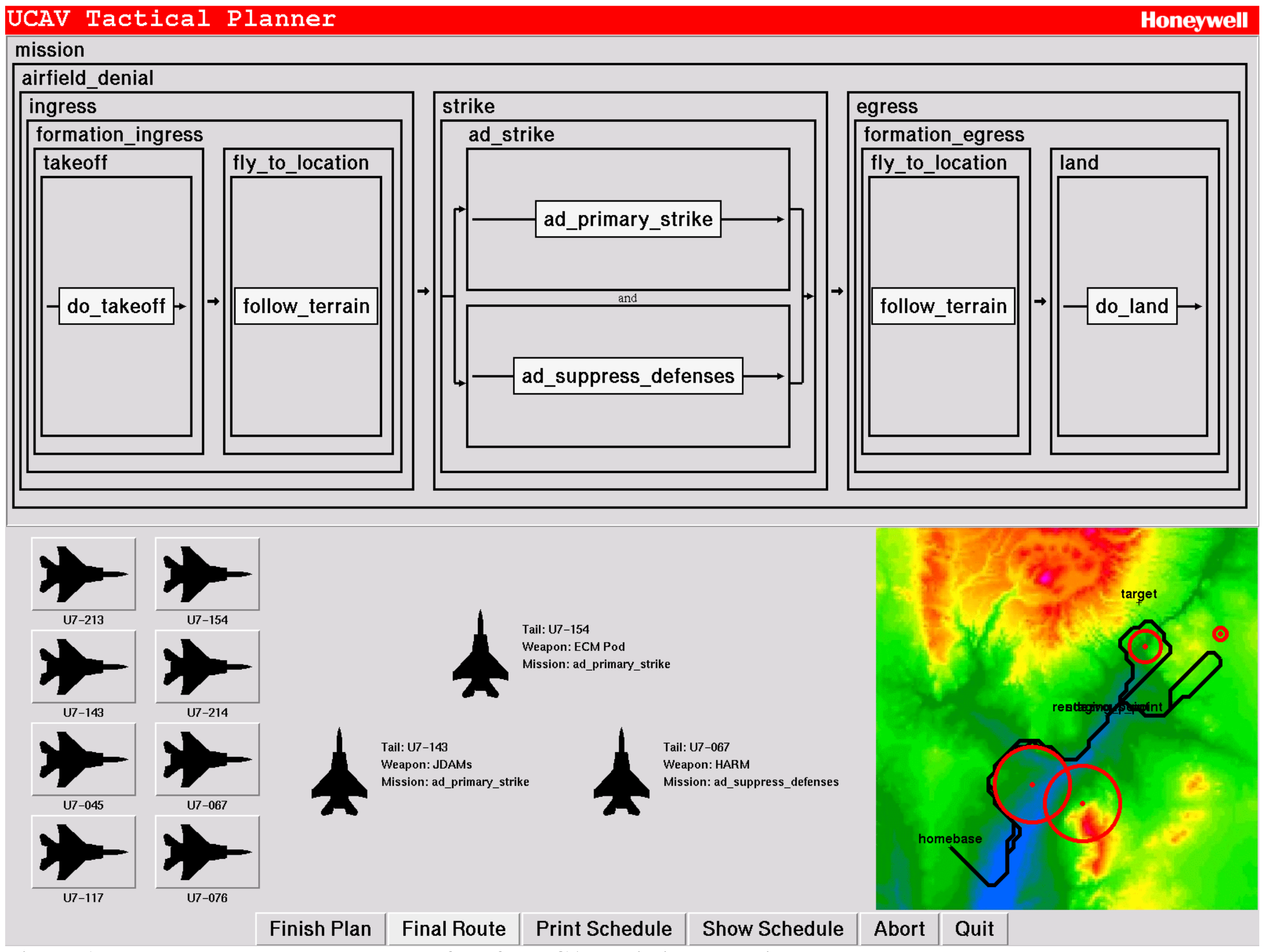


Figure 2. General Playbook Architecture.

Shared Task Model. The operator communicates instructions in the form of desired goals, tasks, partial plans or constraints, via the UI, using the task structures of the shared task model. The MAC is an automated planning system that 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 planning tools (e.g., an optimizing path planner) to perform these functions, wrapping them in its task-sensitive environment. Outside of the tasking interface, but essential to its use, are two additional components. An Event Handling component, itself a reactive planning system capable of making momentary adjustments during execution, takes plans from the Playbook. These instructions are sent to control algorithms that actually effect behaviors.

Operator interaction with the Playbook can be via a variety of user interfaces customized to the needs of the work environment, but operator commands are ultimately interpreted in terms of the Shared Task Model. To date, we have developed prototype playbooks for Unmanned Combat Air Vehicle (UCAV) teams (Miller, Pelican, Goldman, 2000), and Tactical Mobile Robots (Goldman, Haigh, Musliner, Pelican, 2000), and prototypes for the RoboFlag game (Parasuraman, Galster, Squire, Furukawa and Miller, in press) and for real-time interaction with teams of heterogeneous UMVs (Miller, Funk, Goldman and Wu, 2004; Goldman, Miller, Wu, Funk and Meisner, 2005). Below, we provide a description of user interaction with one playbook interface we developed with Honeywell Laboratories to illustrate the general concept.

We developed the playbook illustrated in Figure 2 to enable a human leader to create a full or partial mission



Tail: U7-154
Weapon: ECM Pod
Mission: ad_primary_strike

Tail: U7-143
Weapon: JDAMs
Mission: ad_primary_strike

Tail: U7-067
Weapon: HARM
Mission: ad_suppress_defenses

Figure 1. Prototype Playbook User Interface for UCAV Mission Planning.

plan for UCAVs. This initial work was intended as a ground-based tasking interface to be used for a priori mission planning, but current Playbook work is exploring interface modifications to enable real-time and in-flight tasking and task performance monitoring as well.

Figure 2 shows five primary regions of this Playbook UI. The upper half of the screen is a Mission Composition Space that shows the plan composed thus far. In this area, the operator can directly manipulate the tasks and constraints in the plan. 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 here where they can be interacted with in more detail. Finally, the lower set of control buttons is always present for interaction. 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: Interdiction, Airfield Denial, and Suppress Enemy Air Defenses (SEAD). The mission leader would interact with the Playbook to, first, declare that the overall mission “play” for the day was, say, “Airfield Denial.” In principle, the user could define a new top-level play either by reference to existing play structures or completely from scratch, but this capability has not been implemented yet.

This action is an example of type 2 delegation—providing a specific task for subordinates to perform. But because this is a very high level task in a hierarchical task network, the supervisor has left a great deal of freedom to the subordinates (in this case, the MAC and the UAVs themselves) to determine exactly how a “Airfield Denial” mission is to be performed. If this were the only delegation information the supervisor provided, the subordinates would be obligated to do their best to perform that action (an Airfield Denial mission), but would have a great deal of authority as to how best to accomplish it.

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 tasking interface (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 of the tasking interface is required to provide more information. Here, the human must provide four additional items: a target, a homebase, a staging and a rendezvous point. Each of these is a stipulation, or positive constraint, telling the subordinates that whatever specific plan they come up with to accomplish the higher level mission must include these attributes—and thus, they are examples of type 4 delegation interactions. Most of these activities are geographical in nature and users typically find it easier to specify 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 Playbook UI to provide deeper layers of task selection, or to impose more stipulations on the resources to be used, waypoints to be flown, etc. For example, 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 s/he 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 play structures in the play library, filtered for current feasibility.

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 2). The supervisor does not have to tell intelligent subordinates this; it is a part of their shared knowledge of what an ‘Airfield Denial’ task means—and how it must be performed. To provide detailed instructions about how to perform the Ingress task, however, the user can choose it, producing a “generic” Ingress task template or “play”. 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 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. One of our current challenges in creating Playbooks^{TM} for real-time interactions is to enable them to be sensitive to the current state of affairs and of task performance so as to make intelligent assumptions about task performance possible—for example, if the supervisor wishes to command a currently airborne UAV, perhaps in a holding pattern, to perform an 'Airfield Denial' mission, both supervisor and subordinate should know that the Takeoff portion of an Ingress task is no longer necessary and should either be eliminated or be shown as already accomplished.

The user can continue to specify and instantiate tasks down to the "primitive" level where the sub-tasks are behaviors the control algorithms (see Figure 1) on the aircraft can be relied upon to execute. Alternatively, at any point after the initial selection of the top level mission task and its required parameters, the supervisor can hand the partly developed plan over to the MAC for completion and/or review. In extreme cases, a viable "Airfield Denial" plan for multiple aircraft can be created in our prototype with as few as five selections and more sophisticated planning capabilities could readily reduce this number. But potentially more important, the operator (like a human supervisor dealing with intelligent subordinates) can also provide more detailed instructions whenever s/he deems them necessary or useful to mission success and in the way s/he sees fit.

This Playbook illustrates delegation interactions 2 and 4 (plans and stipulations). The subordinates' role in these types of interaction are described in the table above—to perform the plan through any set of sub-methods that adhere to the stipulations provided by the supervisor, or to report that this is infeasible. One of the MAC's roles in the above example is to report when it 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). In a real-time delegation system, the MAC will be responsible for continual monitoring of performance to report when world states mean that plan performance is no longer capable of (or likely to) accomplish the user's parent plan (e.g., because of equipment failures, adverse head winds, enemy countermeasures, etc.)

The Playbook architecture is, we believe, also capable of supporting delegation interaction types 1 and 3 (goals and negative constraints) as well. Supporting goal-based delegation interactions would require a slight modification to the shared task representation. Currently, we have used a representation that explicitly includes only hierarchically organized and sequenced tasks (i.e., actions to be performed). Tasks implicitly encode the goals they accomplish, but there are representations (such as Geddes Plan-Goal Graphs—Sewell and Geddes, 1990) that explicitly interleave both plans and goals and a linked hierarchy. Use of such a representation, along with related modifications to the UI and MAC, would enable the supervisor to say, effectively, "Today we're going to achieve a State" (e.g., the destruction of a given airfield) rather than or in addition to, the plan-based representation used above which allows only the issuing of task-based delegation commands (e.g., "Today we're going to fly an airfield-denial mission"). The incorporation of negative constraints into the interaction (delegation interaction method #3), would require a less substantial modification to the Playbook architecture—potentially requiring only a UI addition to enable the supervisor to incorporate negative commands about task types and state parameters (e.g., "do NOT fly through this valley or use this type of munition") and then requiring the MAC to create plans which avoid those negative constraints.

Policy—Delegation via Abstract Policy Statements

The final type of delegation interaction offers the ability to provide priorities between alternate goals and states and to do so more abstractly than the above methods. Sometimes supervisors don't have a single, concrete world state goal in mind, much less a specific plan for accomplishing it. Sometimes supervisors must issue commands well in advance to cover a wide range of largely unanticipatable circumstances. In these cases, the delegation instructions will be less a specific statement of actions to take or world states to be sought or avoided, but rather a general statement of outcomes that would be more or less good or valuable (or, conversely, bad or to be avoided) than others. We refer to the set of such abstract value statements that a supervisor might provide as his or her "policy" for performance in the domain.

We have developed policy-based architectures for two applications: providing commanders' guidance to a resource controller for battlefield network communications (prioritizing communications bandwidth in accordance with the commander's intent—Funk, Miller Johnson and Richardson, 2000), and providing visualization and feedback to dispatchers in upset contexts in commercial aviation (Dorneich, Whitlow, Miller and Allen, 2004). We will describe the first of these below.

A policy statement is an abstract, general, a priori statement of the relative importance or value of a goal state in the domain. In its simplest form, policy provides a method for human operators to mathematically define what constitutes "goodness." Once defined, a policy statement can be treated as a rule and evaluated against a current or hypothetical context—if the rule is true in the context, then the context incurs the "goodness" (or badness) value stipulated by the rule. Alternate contexts (which could be tied to the expected outcomes of alternate decisions) can then be evaluated against each other by examining the set of policy rules that are satisfied or violated and the resulting set of goodness/badness values accrued. A set of individual policy statements can be bundled together, and these policy bundles can be used to flexibly define the priorities that apply in a given situation (priorities can change given different circumstances).

A policy-based delegation system requires at least three components: (1) a representation for specifying the "policy" in terms of the value of various partial world states, (2) a user interface for allowing one or more users to input their policies and, if desired, view results of policy application, (3) a computational framework that allows evaluation of a current situation or hypothetical proposed situation against the expressed policy, and (4) an engine that allows application of the policy either to the control of resource application or to a visualization of sensed data about a current situation or projected data about a future or simulated situation.

In the development of a policy-based delegation system for communications resource usage, we were striving to provide a means for commanders to tell an automated network management system their "policies" for how to prioritize the use of communications bandwidth in order to satisfy the most important requests most fully. Note, however, that "most important" was not a static concept but rather changed across commanders and situations. For this application, we developed a policy representation that allowed commanders to assign, a priori and abstractly, a value to various kinds of communications requests. As communications requests then came in from various field units or operators, they could be matched against the commander's policy statements and a value assigned to each of them. This value was then used by a resource optimizing controller to determine which requests should get network bandwidth with what priority.

This process is conceptually illustrated in Figure 3. Each commander's policy is created as a set of statements (as illustrated at the top of the figure) each of which assigns an importance (or value) function to a defined sub-region in a multidimensional space. In this case, individual policy statements are illustrated abstractly as defined by the dimensions: owner (Wi) who is the originator of an information request, source (Si) which is the location of the information to be transmitted, destination (Di) which is the destination to which the information is to be transmitted (i.e., a specific machine or IP address, which need not be that of the owner), and description of the information content (Ci) to be transmitted, along with an importance assigned to that policy statement. For example, policy statements might be based on a single dimension ('Requests for weather information [Content] get Importance 0.2') or on a combination of dimensions ('Requests owned by the Zone Reconnaissance task [Owner] for weather information [Content] from Satellite 476B [Source] to 3rd Air Calvary Division [Destination] get Importance 0.8). If the policy element regions are allowed to overlap, then they must be sequenced (typically from most to least specific) to indicate the order of precedence.

	Owner	Source	Destination	Content	Importance	
(	{W1}	{S1}	{D1}	{C1}	.9	)
(	{W2}	{S2}	{D2}	{C2}	.8	)
(	{W3}	{S3}	{D3}	{C3}	.5	)
(	*	*	*	*	.1	)

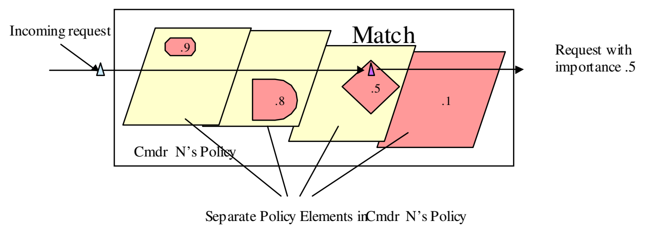

Incoming request
Match
Request with importance .5
.9
.8
.5
.1
Cmdr N's Policy
Separate Policy Elements in Cmdr N's Policy

Figure 3. Representation for a policy for network bandwidth prioritization.

In practice, the commander's policy is then used to assign an importance value to any incoming request for communication resources (illustrated conceptually in the lower portion of Figure 3). Each policy statement defines a region within the multidimensional space defined by the parameters from which policy statements can be built. We've represented that multidimensional space as a simple plane in the figure, and then defined multiple regions within that space with an assigned value for each to represent the valuation contained in each separate policy statement. Each incoming request is matched against the sequenced series of policy statements the commander has made. The first policy element that matches the request determines the importance of that request and informs an automated resource manager about the relative value of satisfying that request.

While conceptually simple, many useful functions can be performed within this framework. First, it is not necessary that importance be construed as an all-or-nothing value as it is depicted in Figure 3. Instead, we have explored more sophisticated representations that allow the requestor to provide a description of how s/he wants the information requested along several dimensions (e.g., freshness, reliability, initiation-time, accuracy, resolution, scope, etc.) Then the resource management system can treat the importance value as a maximum number of value "points" to be awarded for satisfying the request perfectly, while still awarding itself points for partial satisfaction. This permits more sensitive management of resources to be performed.

Second, it is rarely the case that a single commander or supervisor is the only one who may have an interest in dictating policy about how subordinates behave. Rather, each commander must allocate his/her resources in accordance with the policies of those above. We support this requirement (Figure 4) by modeling policies that exist at nodes in a command hierarchy. As requests come in, they are matched against the commander's policy that governs them (command node N1.2.2 in Figure 4), but must then also be matched against his/her commander's policy (i.e., the command node in charge of node N1.2.2 in

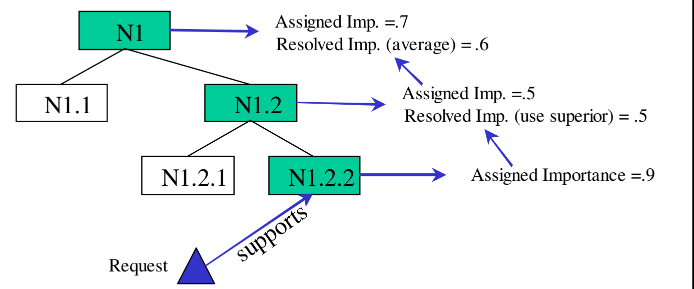


Figure 4. Applying and resolving policy across echelons in a command hierarchy.

Figure 4—node N1.2)—and so on, up the chain of command (i.e., node N.1 in Figure 4). We allow each commander to stipulate how this matching policy element should be resolved with the subordinate commander's matching policy element: as a ceiling or floor value, or linear or weighted combination of the values. Even when a single well-defined chain of command does not exist the policies of different "interest groups" may be represented with relative weights on the importance values that each would assign to a potential outcome. We have used this approach (Dorneich, et al., 2004) in representing the many, varied interests which impact the decisions of a commercial airline's dispatch operators (e.g., crew scheduling, maintenance scheduling, marketing, passengers, finance, etc.).

Folding this policy-based form of delegation interaction (method 5) into an overall architecture that includes the other methods is not as difficult as it might first appear. While we have not yet developed a system that accomplishes this, the way forward is clear. Policy is simply an assignment of value or priority to the goal states and tasks in the other delegation interaction types. Priorities for resource usage and the desirability of various outcomes stem, after all, from a superior's goals and plans for subordinates (whether human or machine). If, for example, I task a given unit under my command to perform an Airfield Denial task, and I know that their task is the most important of all concurrent tasks to me, then I have effectively said that giving them the resources they require to perform that task (specific aircraft, munitions, fuel, communications bandwidth, etc.) represents the highest value to me. In other words, delegation interactions that provide specific goals, plans, stipulations and constraints to subordinates carry with them specific policy implications. Whenever a commander can provide more specific delegation instructions, this will generally get him/her closer to the results desired from his/her subordinates, but this will not always be the case. Hence, the ability to stipulate more abstract policies should probably be preserved in a complete delegation system as a means of covering unexpected and unfamiliar situations.

Conclusions and Future Work

While the work described above represents a general framework for delegation interactions suitable for human interaction with smart automation of various kinds and, perhaps uniquely, suitable for the tasking of multiple UMVs, our work has thus far progressed only to the proof of concept stage. As noted above, we have currently implemented only portions of the various methods of delegation that a fully flexibly delegation interface might benefit from, and have done so in disparate systems. Furthermore, our proof of concept implementations have not yet afforded us the opportunity to do rigorous human in the loop evaluations to demonstrate improved performance, if any.

These situations are changing, however. We are currently engaged in exploration of human interaction with Playbook -like interfaces (Parasuraman, et al., in press) and are performing work on a Playbook interface for real-time interactions with heterogeneous UMV assets by operators who may be concurrently involved in other critical tasks (under a DARPA-IXO SBIR grant-- cf. Miller, et al., 2004; Goldman, et al., 2005). One of the goals of this work will be to develop task libraries and task construction tools and interface concepts to move the delegation interface work along toward implementation and utility.

Of course, anyone who has worked with a poorly trained, or simply mismatched, subordinate is well aware that it is possible for delegation to cause more work than it saves. Our challenge, and that of others who adopt a delegation framework for human interaction with complex and largely autonomous automation, will be to ensure that this does not happen--through judicious use of technology and substantial usability analysis and testing. On the positive side, however, we benefit from the knowledge that delegation approaches to interaction with intelligent yet subordinate actors have worked repeatedly throughout history and, particularly, the history of warfare. As automation in the form of UMVs increasingly takes its place as one of those actors we want to be intelligent, capable and effective yet remain subordinate, we will increasingly need methods for enabling it to interact with us in the ways that we trust and are familiar with. Since delegation is the primary method that fits that bill, it only makes sense to pursue delegation approaches to human interaction with automation.

Acknowledgements

An earlier version of this paper was presented at the NATO Research and Technology Organization Special Workshop on Uninhabited Military Vehicles, June 10-13, 2003; Leiden, Holland.

Many people have contributed to the ideas presented above. A partial list includes Robert Goldman, Harry Funk, Michael Pelican, Dave Musliner, Karen Haigh, Michael Dorneich, Stephen Whitlow, John Allen, and Jim Richardson.

References

Dorneich, M.C., Whitlow, S. D., Miller, C. A., Allen, J. A. 2004. A Superior Tool for Airline Operations. *Ergonomics in Design, 12*(2). 18-23.

Funk, H., Miller, C., Johnson, C. and Richardson, J. 2000. Applying Intent-Sensitive Policy to Automated Resource Allocation: Command, Communication and Most Importantly, Control. In *Proceedings of the Conference on Human Interaction with Complex Systems*. Urbana-Champaign, Ill. May.

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

Goldman, R., Miller, C., Wu, P., Funk, H. and Meisner, J. 2005. Optimizing to Satisfice: Using Optimization to Guide Users. In *Proceedings of the American Helicopter Society's International Specialists Meeting on Unmanned Aerial Vehicles*. January 18-20; Chandler, AZ.

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

Miller, C., Pelican, M. and Goldman, R. 2000. "Tasking" Interfaces for Flexible Interaction with Automation: Keeping the Operator in Control. In *Proceedings of the Conference on Human Interaction with Complex Systems*. Urbana-Champaign, Ill. May.

Miller, C., Funk, H., Goldman, R. and Wu, P. 2004. A "Playbook" for Variable Autonomy Control of Multiple, Heterogeneous Unmanned Air Vehicles. In *Proceedings of the 4th Conference on Human Performance, Situation Awareness and Automation.* Daytona Beach, FL; March 22-25.

Parasuraman, R., Galster, S., Squire, P., Furukawa, H. and Miller, C. In press. A Flexible Delegation-Type Interface Enhances System Performance in Human Supervision of Multiple Robots: Empirical Studies with RoboFlag. Accepted for inclusion in J. Adams, guest ed. *IEEE Systems, Man and Cybernetics—Part A, Special Issue on Human-Robot Interactions*.

Sewell, D. and Geddes, N. 1990. A plan and goal based method for computer-human system design. In D. Diaper, ed. *Human-Computer Interaction—INTERACT '90*. Elsevier Science; North-Holland. pp. 283-288.