Delegation in LoA^3 Space

**SIFT Technical Report
08-LOA3-01**

Mr. Harry B. Funk
Dr. Christopher A. Miller

Smart Information Flow Technologies
211 First St. N., Suite 300
Minneapolis, MN 55401

{hfunk, cmiller}@sift.info
612-339-7438

Identification and Significance of the Problem or Opportunity.

Overview

Problem

There are a plethora of UAV interfaces these days, ranging from conceptual prototypes to fielded systems. The cost of getting these interfaces 'wrong' is high – current UAV losses are occurring at unacceptable rates (over 100 times the rate for manned aircraft—Bone & Bolkcom, 2003). Human factors are already cited as the most common cause of UAV accidents, at least in existing Army reports (Manning, et al., 2004). Similarly, half of U.S. Marine UAV accidents were attributable to human factors (Fergusson, 1999). Although specific data on Air Force platforms are not available to us, it is reported (CNN, 2002) that, since 1994, over half of the Predator R-1 fleet has crashed or been destroyed by enemy fire, with the majority of those occurring during testing and, therefore, not subject to enemy actions. Errors that cited decision making and unsafe supervision are already involved in a greater proportion of UAV accidents than are the lower level perceptual and skill based factors which increased autonomy may assist. Note too that these figures apply only to accidents; they say nothing of less severe forms of suboptimal performance of the human + machine system. Even if the interface is usable, the Army's current vision of employing UAVs as automated "wingmen" to support the crew of attack or scout helicopters requires that the human interaction with these vehicles impose substantially *less* workload than is currently the case for any UAV. In short, the need to understand and design for the human role in supervising and delegating to teams of highly-capable, yet truly *semi*-autonomous machine agents has never been greater.

Solution to Date

SIFT researchers pioneered (Miller & Goldman, 1997; Miller, Pelican & Goldman, 2000) a human-automation integration architecture called Playbook®, in which humans and automation share a task model – the possible ways to achieve an outcome, and what everyone is supposed to do in the play. The user can select - either at a very high level, or if time permits and circumstances require, at much finer levels – how the tasks will be performed, and by whom. Thus, Playbook allows precise control over delegation of tasks to automation. We maintain that Playbook can be used as a testbed for examining different approaches to supervisory control.

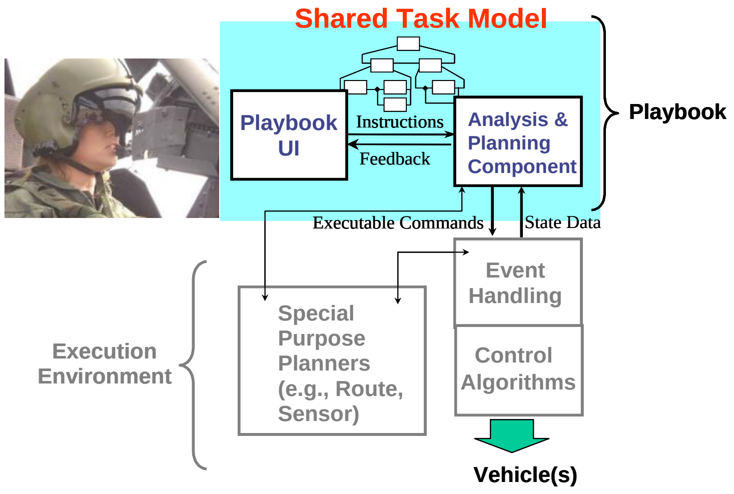


Figure 1. Playbook Architecture for Tasking Interfaces

We developed a theory of Levels of Autonomy, expanding earlier work by Sheridan and Verplank (1978), and Parasuraman, Sheridan and Wickens (2000) to create a three dimensional model for characterizing levels of autonomy, where the dimensions are ***A***uthority X ***A***bstraction X ***A***ggregation – the LoA^3. The authority dimension captures how much independence the automation has with respect to decision-making – e.g., the need to ask permission vs. informing the user vs. full independence. Abstraction captures at what level the automation is tasked – higher, more abstract tasks vs. finer-grained, more concrete tasks. Aggregation captures how much of a resource is tasked at a time – e.g., single platform vs. flight vs. swarm. The intuitively reasonable claim is that there is a 'sweet spot' within this LoA3 space that will result in optimized performance of the human+automation system with respect to various metrics of interest: operator workload, situation awareness, appropriate reliance, and of course, mission outcome.

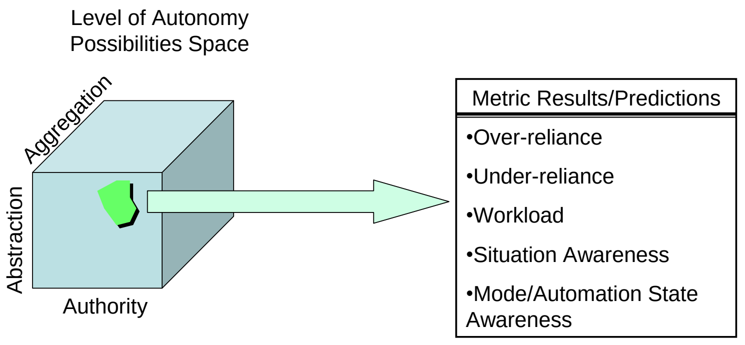


Figure 2. Selection and use of a region in LoA3 space results in a predicted outcome against various metrics that can be verified via simulation.

Playbook supports operation in any region of LoA3 space. That is, the task model can be defined to allow command of the automation with any level of authority, at any level of aggregation, at any level of abstraction in the hierarchy. To make this an experimental testbed, it is simply necessary to allow the user – either the experimenter or the operator – to select the region in which s/he wants to operate. For the experimenter, this may involve 'lopping off' sections of the space to constrain the operator interactions and observe the results (as compared to another set of constraint conditions). For the operator, this involves selection of what s/he considers the best interaction with automation. Both of these user interactions provide information as to the 'appropriate region of LoA3 space.

This begs the question, "Appropriate in the face of what conditions?" We expanded the concept to include a predictive mapping from what we believe are the core relevant factors to the appropriate region in LoA3 space. The testbed we propose will allow experimental verification of our predictive model via simulation. At the completion of Phase II, for the first time we will have a comprehensive theory supported by experimental results that shows how best to support our operators in their complex operational environments. Better still, the model can be used to dynamically alter the LoA3 space that the operator should use, as conditions change. Perhaps best of all, the move from experimental testbed to operational product is straightforward – the testbed environment directly uses the operational tool.

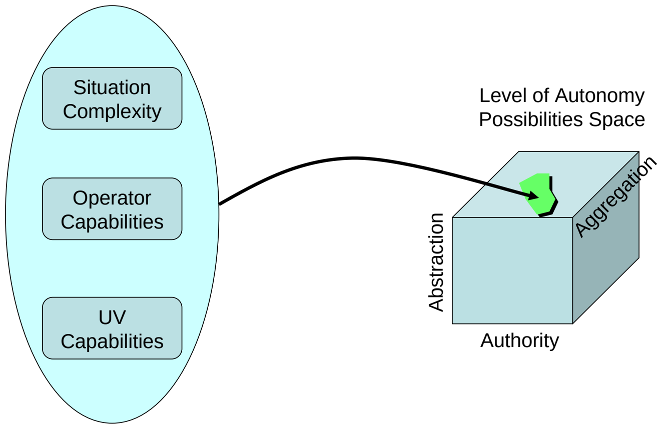


Figure 3. Given measures for the complexity of a situation, an operator's competence, and capabilities of the UV platform(s), we can predict the appropriate regions of LoA3 space.

We have identified the necessary modification and extensions to Playbook and created an initial prototype that shows promising results. The remainder of this proposal will detail the underlying theory, results to date, and plans to achieve this vision.

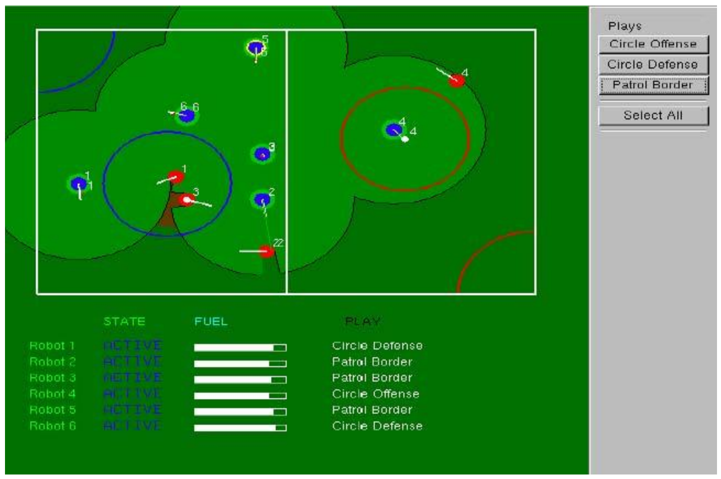
Theory of Delegation Interfaces

Our design for the supervision of multiple UAVs/UGVs is based on the concept of *delegation* (Miller and Goldman, 1997; Miller & Parasuraman, 2006) and implemented in SIFT's Playbook® architecture (Miller, 2003; Miller, Funk, Goldman, & Wu, 2003). Previous work on interaction between humans and automated agents, including unmanned vehicles, has revealed both benefits and costs of automation for system performance (Parasuraman & Riley, 1997; Parasuraman, Sheridan, & Wickens, 2000; Sheridan & Parasuraman, 2006). Automation is clearly essential for the operation of many complex human-machine systems. But in some circumstances automation can also lead to novel problems such as increased workload and training requirements, impaired situation awareness and, when particular events co-occur in combination with poorly designed interfaces, accidents (Degani, 2004). Retaining the benefits of automation while minimizing its costs and hazards may require the interface between humans and robotic agents to be adaptable, rather than fixed and static. The performance benefit of adaptive compared to static automation is well documented (Parasuraman et al., 2000). However, if adaptation is executed without user approval or knowledge, the cost of system unpredictability may outweigh the benefit that automation provides. What is needed is a method that allows operators to explicitly task automation at times and in the fashion of their choosing—as in delegation to a human subordinate—using an interface that incorporates high-level communication between human and automation in a common language. Delegation-type interfaces represent this form of adaptable automation (Miller et al., 2003; Miller & Parasuraman, 2006).

The key idea behind delegation in human-human contexts is that the supervisor generally has flexibility in the use of supporting agents. The operator can delegate bigger, coarser-grained tasks or smaller, more precise ones with more or less explicit instruction about their performance, depending on context and task demands. Delegation architectures for human-automation interaction seek to provide highly flexible methods for the human supervisor to declare goals and provide instructions and thereby choose how much or how little autonomy to give to automation, depending on context and the current situation.

Experimental Validation of the Delegation Interface Concept

Prototype delegation systems for a variety of UV and other military applications have been developed (Miller et al., 2003; Parasuraman and Miller, 2006). While prototype development is valuable, we believe that any interface concept must be validated experimentally in human-in-the-loop simulation studies (Miller and Parasuraman, 2007). Our team has begun this validation for the delegation interface design concept in previous experiments using a high-fidelity simulation—termed RoboFlag—of multiple autonomous, mobile robots engaged in a game of "capture the flag" (Parasuraman et al., 2005).



	STATE	FUEL	PLAY
Robot 1	ACTIVE		Circle Defense
Robot 2	ACTIVE		Patrol Border
Robot 3	ACTIVE		Patrol Border
Robot 4	ACTIVE		Circle Offense
Robot 5	ACTIVE		Patrol Border
Robot 6	ACTIVE		Circle Defense

Figure 4. RoboFlag delegation interface

The simulated robots in RoboFlag move and navigate autonomously, exhibit cooperative behaviors, and act as a team with other robots to pursue the goal of capturing the opposing team's flag (Figure 4). Parasuraman et al. (2005) modified the simulation to emulate a generic mission involving a single operator managing a team of UVs. The mission goal was to send the robots from the home area into opponent territory, which was populated with a red team consisting of the same number of robots. The target flag had then to be accessed, picked up, and returned to home territory without being tagged by the opposing robots. Individual human operators supervised up to eight robots using a simplified delegation interface. The interface allowed the operator to use automated behaviors or "plays" (Miller et al., 2003), as well as manual (waypoint) control, in pursuing the mission goal. Plays directed robots to engage in autonomous, cooperative behaviors in pursuit of a sub-goal; e.g., in "Patrol Border", a user-selected number of blue team robots went to the border area, spaced themselves appropriately along the length of the border, and maneuvered to prevent red team robots from crossing.

One of the postulated benefits of delegation-type interfaces (Miller and Parasuraman, 2007) is that they allow for flexible use of automation in response to unexpected changes in task demands, while keeping the operator's mental workload in managing the automation within a reasonable range. To test this hypothesis, Parasuraman et al. (2005) varied two sources of task demand: (1) the adversary "posture," in which the opponent engagement style was changed unpredictably to be offensive, defensive, or mixed; and (2) environmental observability, in which the effective visual range of each robotic vehicle under the control of the operator was varied from low to high. Measures of performance included mission success rate, mission execution time, strategy use (plays vs. manual control), subjective mental workload, and global situation awareness.

The results confirmed that the delegation interface allowed for effective user supervision of robots, as evidenced by the number of missions successfully completed and time for mission execution. In the most challenging condition (mixed posture with opponent uncertainty), users had moderate success (about 62%) and relatively short mission completion times. These findings suggest that the delegation interface allowed users to respond effectively to unpredictable changes in opponent posture by tasking robots appropriately. A second experiment conducted by Parasuraman et al. (2005) confirmed these results and also showed that the benefits resulted from the flexibility afforded by the delegation interface as implemented in RoboFlag. Specifically, the flexible delegation approach was compared to fixed delegation approaches, by providing users: 1) only manual control or 2) only automated plays or, 3) both types of control, under the same varying adversary postures (offensive, defensive, mixed) manipulated in the first experiment. It was hypothesized that the use of the delegation interface would afford users maximum flexibility, allowing them to decide when and how to use automation (e.g., when workload was and the automation was effective). The mission performance measures supported this hypothesis. Additionally, the flexible delegation interface allowed users to mount a more effective response to variable opponent postures than did the static control conditions (manual or automated).

These results point to the potential for delegation interfaces to enhance mission effectiveness in multi-robot supervision by a single operator. But it has illustrated utility in allowing for effective supervisory control of multiple numbers of UVs, we have examined only a few of the dimensions of along which delegation control might vary and, of course, our (simulated) robotic capture-the-flag players are a far distance from real applications. Additional work is needed to further validate the delegation interface concept, much less to apply it to complex, real-world, high-criticality domains such as military operations. The potential cost of delegation interfaces in terms of the workload required to instruct automation also needs additional examination with objective as opposed to only subjective measures of workload. Finally, delegation interfaces and application domains with a greater range of capabilities than the simplified interface used in the RoboFlag studies need to be evaluated—especially delegation interfaces and delegation scenarios appropriate to Army aviation domains.

Background

Our objectives for Phase I of this work were to use an innovative concept of the "space" of possible delegation interactions in an existing delegation tool to both formulate experimental manipulations of interest *and* to serve as the controlling architecture for an experimental testbed to enable those experiments. In this section, we describe necessary background information, developed under prior work, for understanding our approach to this project: this includes: (1) the existing delegation tool (Playbook), (2) the "delegation space" concept (LoA^3) and its role in characterizing dimensions along which different human-automation relationships may vary, and (3) our general concept for using Playbook and the LoA3 architecture to serve as an experimental testbed for researching the effectiveness of alternate human-automation relationships in this project..

Playbook Description and Status

SIFT has pioneered a human-automation integration architecture, called Playbook®, based on a shared model of the tasks in the domain (Miller & Goldman, 1997; Miller, Pelican & Goldman, 2000; Miller, Funk & Goldman, 2004; Miller, 2005; Miller & Parasuraman, 2006; Parasuraman & Miller, 2006, Miller & Parasuraman, 2007). The task model provides a means of communication about plans, goals, methods and resource usage—a process akin to referencing plays in a sports team's playbook. A play can be "called" very quickly in a variety of contexts and then adapted to the current situation through the intelligence resident in the "system" (i.e., for a sports team, the players themselves; for a team of UVs, either in the UVs or in a ground control station). Alternatively, a coach can adapt, refine or specify a play with minimal effort (since everyone knows the basic play "vocabulary" to begin with) if time permits or the situation requires. SIFT's Playbook is designed to enable human operators to interact with subordinate systems with the same flexibility. For Unmanned Vehicles (UVs), Playbook actively manages missions but enables the human to "call plays" at either a very high level, or to progressively "drill down" in the task decomposition to refine the way in which a particular instance of a play is to be carried out this time. At whatever level the supervisor decides to delegate instructions to the UVs, Playbook then autonomously develops plans to accomplish those instructions within the user's constraints. If no such plan is feasible, Playbook states that as feedback and, in our more recent implementations, offers a close alternative that is feasible.

The basic Playbook architecture is illustrated in Figure 1 and uses a task model to share information between the user, the user interface (UI) and a constraint-based planning engine (the Analysis and Planning Component—APC). The task model is both hierarchical and sequential. When a task (or, approximately synonymously, a *play*) is activated, the model provides the system knowledge of all the required and optional methods that may be used to accomplish it. The human operator expresses a more or less detailed command via the UI and the APC fleshes it out to an atomic, executable level for the devices to be used (e.g., UAVs). Once an acceptable plan of atomic actions is created, it is passed to an Event Handling component, an outer-loop control system capable of making in-flight adjustments to the plan. The Event Handling component sequences control algorithms that actually effect behaviors in the (possibly simulated) controlled vehicles.

Shared Task Model and the Meaning of a Play

Correctly capturing the semantics of plays is critical to acceptance of the Playbook. Plays are, in essence, compressed commands and, when issued, the requestor has an expectation that the action taken will conform to his/her idea of the request's meaning. The Shared Task Model forms the 'language' through which humans and automation communicate about task performance. The Shared Task Model is a hierarchical decomposition of tasks that captures the functional relationships and sequential constraints between labeled 'packets' of goal-directed behavior. A 'play' is an encapsulated set of behaviors—perhaps including alternatives—that provides a method of accomplishing domain goal(s). Plays are not scripted, static procedures, but rather dynamic templates that identify a range of behaviors. Because plays (and their component tasks) are hierarchically defined, a very complex suite of behaviors can be efficiently referenced by asserting the label of the parent task. Plays are largely synonymous with tasks, though they may represent a specific aggregation of lower level tasks with a defined label—a macro operator that can be adapted to the context in which it is 'called'. Even when modifications to existing plays are necessary, or whole new plays need to be created, we can do this very quickly by modifying existing plays.

Figure 5 shows possible expansions of a task we have implemented in prior work: **`Prosecute Target`**—tracking, designating and firing upon a target, with subsequent Battle Damage Assessment (BDA). Each of the subordinate tasks itself is defined in terms of the subtasks that must be performed to complete the parent task. This definition process repeats until an atomic task level is reached. Atomic tasks are those that are executable by a given platform.

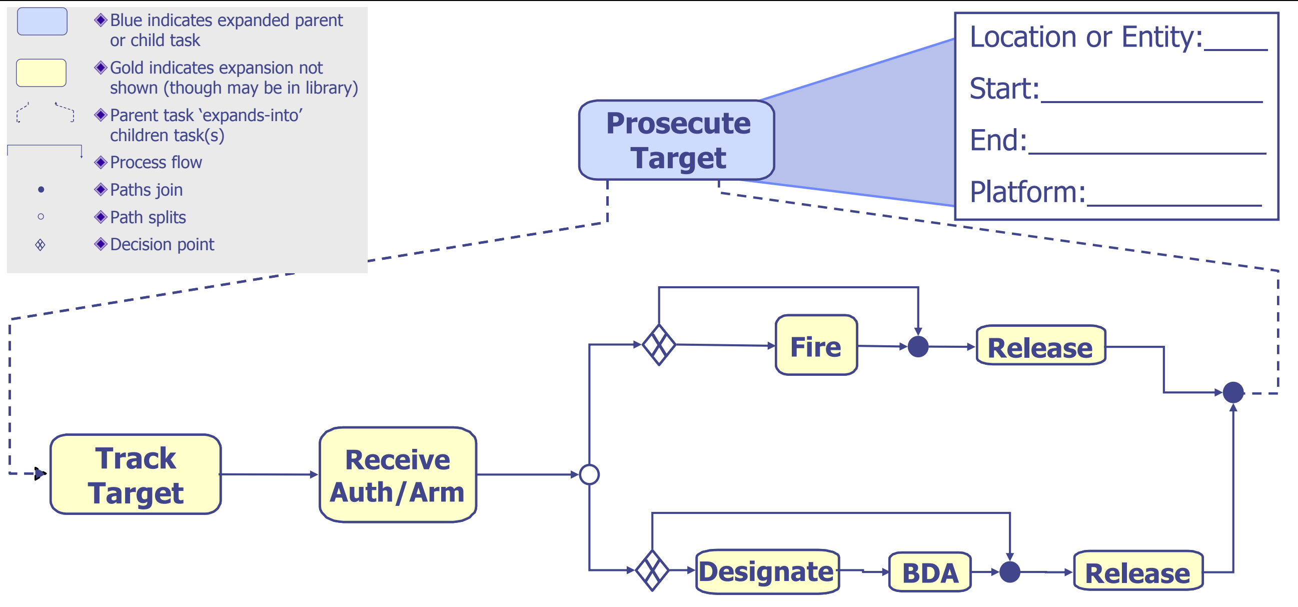


Figure 5. Play Example – Prosecute Target.

Task templates at all levels contain variables that must be set. Some parameters must be specified by the human operator for the task to be executable. Others can be set by either human or by the APC. Still others, generally at the lower levels of the task hierarchy such as flight control settings, can only be set by the APC or by real-time control algorithms themselves. When the APC sets a parameter, it does so within constraints imposed by the task template, by the user, and by the need to develop a task series which accomplishes the goal(s). Automated parameter settings take their values from the current context, have reasonable default values passed from other plays, or are set according to heuristics for efficiency, safety and likely plan success.

In the multiple prior implementations of the Playbook concept (see **Error! Reference source not found.** below), we have used a variety of representations to encode play knowledge. Thus, we are familiar with a variety of potential approaches, including object-oriented styles and even Prolog representations. Ongoing work for a parallel, DARPA-funded program, Integrated Learning, is substantially revising and enhancing the capabilities of the underlying planning engine and representation.

Analysis & Planning Component

The Analysis & Planning Component (APC) is a central part of what we term 'Playbook'. The core of the APC is an enhanced version of a hierarchical task network planner known as SHOP2 (Nau, et al., 2004). The APC evaluates the feasibility of alternate methods of satisfying requested plays. When given a high-level play request, the APC selects among various hierarchically organized methods to compose a feasible plan, then issues instructions to the execution environment (either simulation or UV platform) and monitors for necessary revisions during performance. When given lower-level, more specific and detailed requests (such as a specific platform to use, paths to follow, or scan patterns), the APC reviews them for feasibility and either (a) reports if requested actions are infeasible (and enters into a negotiation about what modifications the user might be willing to make), (b) passes 'validated' user requested plans to the execution environment and monitors their performance, or (c) fleshes out operator requests to an executable level. The APC is designed to use special purpose planners as adjuncts to its planning process; the route planner currently in use for one of our programs is a GIS package modified to perform the route plan function.

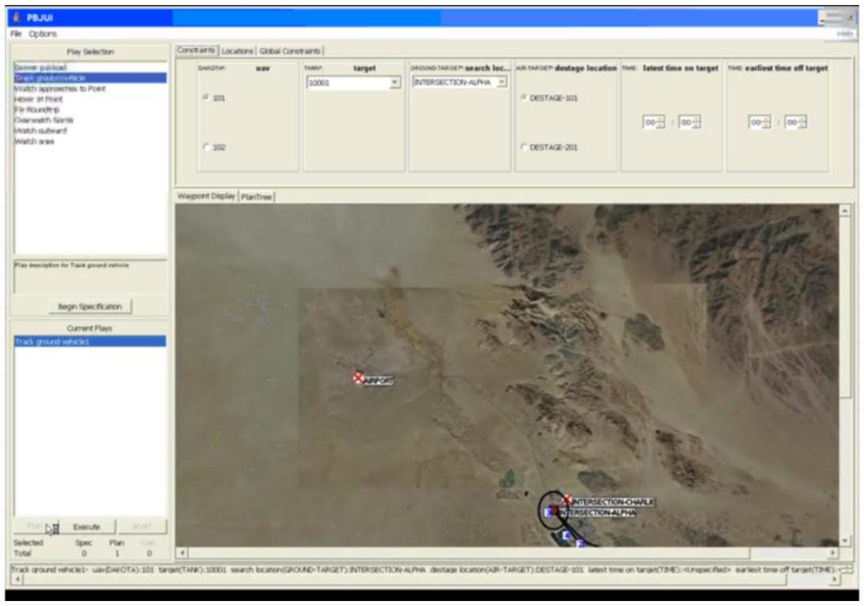
User Interfaces

The Playbook architecture, with its shared task model, can support a wide variety of user interfaces, customized to different work domains. Robust interaction with the shared task model, suitable for detailed tasking and/or negotiation over the tasks to be performed, might be suitable for pre-mission planning or mission control applications where the human has substantial time, minimal distractions and access to a large amount of display space. An interface designed to support the creation or modification of novel plays would require both substantial time and space for the operator, as well as sophisticated tools for manipulating and storing play components—and would benefit from simulation components to examine the realization of the newly-defined play in alternate contexts. By contrast, a playbook application designed for use in a time- and space-constrained domain, perhaps during real-time execution of multiple concurrent tasks (such as UAV control by an aviator concurrently flying his own aircraft), would likely need to make use of a reduced number of highly "compiled" plays, perhaps with a few specifiable parameters, that the user could select or, perhaps, literally "call" (via voice actuation) while heavily loaded with other tasks. Note that in each case above, the user is manipulating the underlying task structure of the plays and communicating via that structure to the APC.

To date, we have implemented a total of five partial prototypes of the Playbook system, each with a slightly different user interface to meet the needs of a slightly different domain of use. The first four are described in Parasuraman and Miller (2006), and the latest (funded under funding from the Army's Unmanned System Initiative) is shown as Figure 6. These UIs serve to illustrate the range of interface possibilities using a Playbook approach.

Playbook Benefits

Playbook, and delegation-style interfaces in general, are postulated to overcome many difficulties associated with highly automated systems (see Miller and Parasuraman, 2007, for a review). This is by virtue of their flexible, user-directed interaction style which requires the user to maintain enough situation awareness to make informed decisions about the level of automation support to use, and to provide specific instructions to provide to that automation at any point in time. These problems include the well-documented (see Parasuraman and Riley, 1997, Parasuraman, and Byrne, 2003 and Billings, 1997 for reviews) problems of loss of skills, loss of situation awareness, mis-tuned trust, unbalanced workload, lack of user acceptance and sub-optimal overall human + machine system performance.



Track ground vehicle1- uav(DAKOTA):101 target(TANK):10001 search location(GROUND-TARGET):INTERSECTION-ALPHA destage location(AIR-TARGET):DESTAGE-101 latest time on target(TIME):<Unspecified> earliest time off target(TIME):

Figure 6. Current PBJUI interface

In previous research described above (Parasuraman, et al., 2005) we obtained initial empirical evidence for the efficacy of delegation interfaces. These findings provide support for the view that the delegation interactions allow for effective tasking of multiple robots while keeping the operator in the decision-making loop, without increasing operator mental workload, and allowing the human operator to adapt successfully to unpredictable changes in the environment. These benefits are important because traditional human-automation interfaces have often been found to result in significant system and human performance costs—including mode errors, user under- and over-reliance on automation, and reduced situation awareness (Parasuraman & Riley, 1997; Parasuraman, Sheridan, & Wickens, 2000). Such limitations are sometimes severe enough to result in catastrophic accidents, as evidenced by numerous analyses of aviation incidents, including unmanned aircraft such as the Air Force's

Predator (Parasuraman & Byrne, 2003). Hence, the development of appropriate human-automation interfaces is critical for effective human supervision of autonomous agents, including robots and unmanned vehicles. Playbook provides such an interface concept. As Parasuraman et al.'s (2005) prior findings suggest, its benefits may be particularly apparent in situations of environmental uncertainty and where unexpected events occur, which can make pre-programmed automated behaviors ineffective.

Open Questions in Playbook Use

The history of human-computer interaction design is replete with instances of interfaces that were initially thought to be beneficial but were inadequately tested, with the result that they were found to be plagued with subsequent problems (Schneiderman, 1993). Recently, this has hit home in the UAV domain with research showing that higher levels of automation (Management by Exception, as compared to Management by Consent), assumed to be beneficial to the operator, actually degraded human+automation performance (Ruff et al., 2004) It is therefore important that the potential drawbacks of any new interface concept are also investigated, so that a balanced and contextualized view can be drawn of its effectiveness and usability.

While Parasuraman et al. (2005) and a follow-up study by Squires, Trafton, & Parasuraman (2006) provide initial evidence for the efficacy of a delegation style interaction, this work has many limitations. First, it used a simple form of delegating—involving only a single commandable behavior at a time rather than the complex, sequential and conditional behaviors which can be expressed in a Playbook play. Second, it involved a simple and very short term task to be supervised by the human. Third, it did not explore variations in automation reliability, human concurrent workload, prior trust, or any number of other factors known to be important in automation usage decisions and overall performance (Parasuraman and Riley, 1997; Lee and See, 2004).

Finally, although this prior research illustrated that human-machine delegation can provide benefits in some instances, it has not done much to discover the range where such interactions may and may not be beneficial. Other research (particularly Kirlik, 1993) clearly shows that humans sometimes cannot make effective use of flexibility—e.g., when the workload required to decide how to use automation and/or to instruct it exceeds the benefit it provides. The development of useful human-machine delegation will depend on our ability to accurately characterize the trade off space for when and how delegation interactions will provide enhancements to overall performance.

Specific open questions for Playbook use that will motivate our investigations in this proposed work include:

- The effects of various forms of uncertainty on managing assets in a delegation framework
- Effects of task interruptions, especially those occurring during a time-critical tasking interaction
- Effects of concurrent workload—do operators do a good job of managing tasks by their priority when some of those tasks may involve instructing (or putting off) subordinates
- Management effectiveness—when (in terms of skill acquisition) are humans good at tasking? Also, how do transitions work: do effective managers stay effective when they get more or fewer assets?
- Trust in a Playbook/delegation setting—how is it built? Managed? Destroyed? There is substantial work on trust in traditional automation, and some in trust for adaptive automation, but virtually none on trust in automation in which the human user provides flexible instructions and observes performance as a result.

- How can/should performance monitoring be shaped given a set of tasking instructions? What should the supervisor monitor? How can the subordinate know when to inform the supervisor (about deviations)?
- Effects of modalities/styles of communication; uncertainty of communications—while we have explored various user interface concepts for Playbook to date, these have not been systematically varied and their effects on performance under different contexts have not been measured.
- Optimizing the timing of tasking instructions: Does the machine have the right to say "don't bother me"? When does it have the right/duty to ask for further instructions?
- Tradeoff between flexibility and time—under what circumstances is it less useful (in terms of ultimate performance) to have access to greater flexibility if exercising that flexibility incurs greater costs in terms of time and workload? Are humans good at detecting those circumstances?

Levels of automation (LoA) and LoA^3

In order to adequately address these questions, we need a *delegation framework* to structure the various possible implementations – a means of characterizing the appropriate **level(s) of automation (LoA)**.

In order to study alternate forms of delegation relationships and the situations in which they perform well or poorly, it will be helpful to have a "chart of the terrain." In this context, that chart consists of a description of the dimensions along which human-automation task performance and decision making authority relationships can vary. Only then will we be able to characterize the different relationships that are available in different tools or contexts. Only then will we be able to associate performance of a given relationship with a given context.

Fortunately, there have been many previous attempts to characterize dimensions of human-automation relationships. They have generally been referred to as levels of automation or levels of autonomy schema. An LoA scheme is simply a way of dividing up, describing and easily referencing combinations of human and automation behaviors—each of them representing a defined *relationship* between human and automation. A LoA scheme can be used to characterize or describe the mix of human and automation control in a given system—usually in some less-to-more series of "levels". As such it is primarily of interest to theoreticians and, maybe, system designers. If an LoA scheme is to be *used* as a means of controlling a system and achieving work goals, multiple levels must be available to an operator and it must be possible to change or adapt among them on the basis of various aspects of context: user preference, current workload, criticality and expected user performance, etc. In other words, it must be an adaptive automation system. The issue with various schemes for levels of automation is the granularity and sensitivity with which they carve up the range of possible human/automation interactions to control a system.

Table 1. Levels of Automation (after Sheridan and Verplank, 1978).

1.	Human does it all
2.	Computer offers alternatives
3.	Computer narrows alternatives down to a few
4.	Computer suggests a recommended alternative
5.	Computer executes alternative if human approves
6.	Computer executes alternative; human can veto
7.	Computer executes alternative and informs human
8.	Computer executes selected alternative and informs human only if asked
9.	Computer executes selected alternative and informs human only if it decides to
10.	Computer acts entirely autonomously

Sheridan (1987) proposed the best-known framework for LoAs, which we illustrate in Table 1. Sheridan's initial work (Sheridan and Verplank, 1978) offered a one-dimensional spectrum of 10 patterns of human and automation behavior and responsibilities ranging from no automation involvement through increasingly autonomous behaviors from automation, to wholly automatic (no human involvement). This scheme has the advantage of being extremely easy to understand, use and remember, but it is not particularly sensitive and it has been criticized for confounding many different types of automation tasks or information processing functions in its several levels.

Parasuraman, Sheridan and Wickens (2000) realized that this one-dimensional scheme failed to discriminate between some very different uses to which automation could be put. They characterized these uses in a second dimension (see Figure 7) — that of information processing stages or functions. They identified four processing stages: Information Acquisition, Information Analysis, Decision Selection and Action Implementation. They argued that more or less automation support can be provided for each of these functions within a single system and the pattern of high and low automation support for each of these four functions characterizes every aiding system.

We have argued elsewhere (Parasuraman and Miller, 2003; Miller & Parasuraman, 2007) that the two dimensional LoA scheme proposed by Parasuraman, et al., while a major improvement in sensitivity over one-dimensional schemes, remains too coarse-grained for many uses. In particular, this scheme is not adequate to enable rich delegation interactions between human and automation because it doesn't reference *specific* tasks that either entity may have responsibility for. In order to have the same kind of "conversation" about who should do what and how that human supervisors can have with intelligent human subordinates, machine automation needs to understand and be able to reason about the same kind of task-based, hierarchically and functionally organized knowledge structures that humans seem to use.

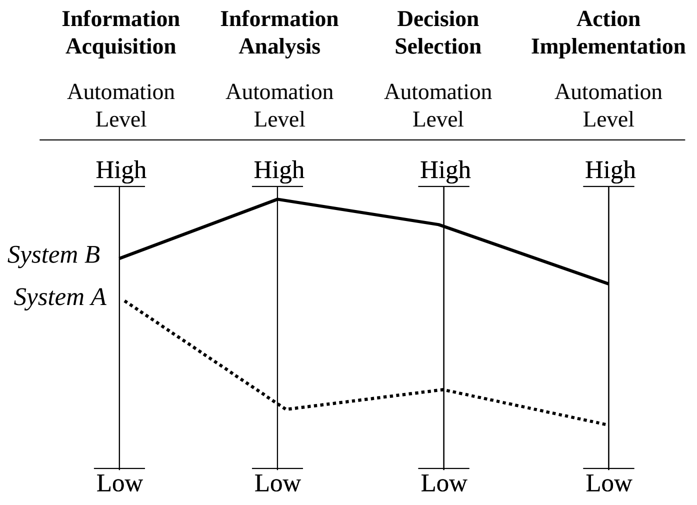


Figure 7. Levels of automation by information processing phase for two systems (from (Parasuraman, Sheridan and Wickens, 2000.)

Through its use of a Shared Task Model, SIFT's Playbook® approach to delegation interactions is one approach that strives to achieve this interaction (Miller, 2003; Miller & Parasuraman, 2007). Playbook may not seem to provide a LoA architecture at first glance, but we would claim that Playbook's task model, combined with its ability to allow a user to specify what automation does and what s/he will do, does the same thing that the prior LoA schemes do: it specifies relationships between human and automation. Playbook does this, however, at a much finer level of granularity than previous schemes. Playbook's task model serves as, essentially, a very fine-grained LoA spectrum, allowing users to say *exactly* what tasks they want automation to do and how rather than relying on setting a coarser-grained LoA . Furthermore, because Playbook's task model is hierarchically structured, users can interact with it at higher or lower levels of functional detail—commanding high level functions or drilling down and making specific stipulations

An LoA scheme of some sort is necessary to characterize differences in possible human-automation relationships and, therefore, whatever LoA scheme we decide to use for research into such relationships should include the dimensions we may wish to vary in experiments to determine which relationships are best in which contexts. Too coarse a scheme will fail to provide the sensitivity to detect important differences. Hence, the definition of an LoA architecture is at the heart of both any delegation approach and any series of experiments or tests designed to evaluate the appropriate range of human-automation interactions for a given context.

Below, we first lay out a rich "three dimensional" LoA framework for human-automation delegation—our LoA^3-- and then show how the Playbook can be used to configure, manage and adaptively command different regions in this "delegation space"—both by an operator at mission planning or even run time, and by an experimenter interested in setting up different experimental conditions for exploring effective combinations of humans and automation in the performance of a range of tasks.

Three Types of LoA

The LoA^3 framework for describing human-automation relationships is based on understanding the dimensions along which such relationships may vary. As described above, previous attempts to provide LoA frameworks have been criticized for confounding these dimensions. LoA^3 provides additional dimensions and finer resolution in describing human-automation relationships than prior LoA schemes provide, while still trying to retain a manageable, useful number of dimensions.

We began the development of the LoA^3 framework by asking ourselves when a human + machine system has *more* automation (or, alternatively, a higher "level of automation"). This seems to be the case when any of three things is true:

1. The automation can perform whatever it does more independently, with more authority; it doesn't have to submit its decisions to a higher authority for approval or review.
2. The automation can be tasked at higher, more abstract levels—that is, it can be relied on to make more of the decisions about how to execute a broader, higher-level task.
3. The automation controls more resources or assets.

These dimensions are not entirely independent. Instead, they provide three views into the delegation relationship between a supervisor and subordinate and they together define a "delegation space" within which achievable relationships can be described.

It is not surprising that prior efforts to characterize Levels of Automation express one or more of these dimensions. Sheridan's levels (as shown in Figure 1 above) say less about specifically what tasks or goals automation is performing, and more about the relationship between the human and the automation. That is, the levels describe the automation in terms of the *authority* relationship between human and system. Does the automation have the authority to do whatever it is it's doing without prior approval? If so, then it operates at Sheridan's level 10. If it can do whatever it does under its own recognizance but must report what it has/is doing to the user, then it is at Sheridan's level 7, etc. This spectrum describes *how* human and automation relate to each other, but doesn't say much about specifically *what* each of them is doing. For convenience, we will refer to this characterization of authority or autonomy relationships between human and automation as the ***Level of Authority*** dimension. Examples of authority levels for a flight task managed by Playbook are shown in Table 2 .

Table 2. Authority level examples

Authority Level	Example
Full	Allows the APC to begin execution of its preferred Overfly plan without either informing or seeking permission from the user.
Inform	Allows immediate execution, but requires informing the user of the selected plan (a capability we already provide, to at least some level of utility).
Override	Allows the APC to begin execution while informing the operator, but allows the operator the ability to jump in and override portions of the APC's plan. A trivial implementation of this capability would be to allow the operator to input novel waypoints or to "grab a joystick" and begin flying the aircraft. The issues, which remain to be clarified, are whether or not it is technically feasible for the operator to override only a portion of the play and allow the rest to remain intact. This is challenging and may not be feasible—the existence of a level of authority in no way guarantees that it is desirable or feasible.
Recommend	allows the APC to present a recommended plan, but requires the operator to manually fly it
Monitor	Involves the APC monitoring the human's plan and execution of it for critical parameter completion such as time to target and fuel usage.

By contrast, Parasuraman, et al.'s two-dimensional scheme adds some description of *what* the automation is doing, albeit in terms of four coarse-grained information processing categories. It begins to describe what each of the actors is doing within a specified LoA—therefore it is a description of activity. We argue that the activity dimension can be subdivided into many finer categories and represented by a hierarchical task model. In other words, the four information processing stages are a coarse categorization of specific tasks. Instead of saying that "information acquisition automation is high" we could more precisely say that automation is deciding how to direct and configure sensors and reporting those decisions to the operator. For a more detailed elaboration of this argument, see Miller and Parasuraman, 2003; and 2007.

Since we can use a hierarchical task model (itself an abstraction hierarchy where higher level tasks are abstractions comprised of their lower level "child" tasks) to characterize what human and automation are doing, we can refer to this dimension of automation activity or behavior as a ***Level of Abstraction***. Automation can have responsibility for higher- or lower-level tasks within the task hierarchy. Having responsibility for higher level (more abstract) tasks presumes responsibility for the tasks which fall below them (although the person doing the delegation retains the right to place constraints or stipulations on the range of decision possibilities about how to perform lower level tasks). So a "Level of Automation" is, therefore, even in Parasuraman, et al.'s model, a combination of a level of authority and a level of abstraction—automation has responsibility for one or more tasks at a given level of abstraction and with a given level of authority.

The Level of Authority x Level of Abstraction framework described above characterizes who is doing what and how they relate to each other. Especially in military domains, however, authority relationships are frequently defined along resource lines as well as task or functional relationships.

Hence, we define a third dimension along which to characterize delegation—a ***Level of Aggregation***[1]. The level of aggregation identifies how much (and/or which type) of resource each actor is authorized to use. When a supervisor delegates a task to a subordinate, that subordinate will be granted authority over some set of resources (including access to his or her own time and energies).

These three dimensions, Level of Authority, Level of Abstraction and Level of Aggregation, therefore define a ***Delegation Space*** of human-automation relationships within which delegation occurs and can be characterized. In short, ***delegation means giving to a subordinate the responsibility to perform a task (with its subtasks), along with some authority to decide how to perform that task and access to some resources with some authority to decide how to use them to perform the task.*** Thus, the three scales must be used to specify four variables which define the delegation space: the level of abstraction and the level of authority on it, and the level of aggregation and the level of authority on it. Note that in most situations, if practical work is to be accomplished by the delegation relationship, the level of authority for the abstraction dimension must be matched by some reasonable degree of authority over resources on the aggregation dimension. Failure to achieve this results in the lament, "I have the responsibility but not the resources." (or the contrapositive, wasted resources).

Figure 8 illustrates these relationships. Imagine a set of tasks which can be performed in a domain, arranged in a hierarchical, abstraction relationship. A supervisor *controls* some portion of those tasks—which means s/he has *authority* over deciding when and how they need to be performed and when and how to use resources (which s/he also controls) to accomplish them. When the supervisor delegates some of those tasks, s/he is delegating that control—over the decision(s) about how and when they need to be performed, and over the decision(s) about how and when to use allocated resources to accomplish them.

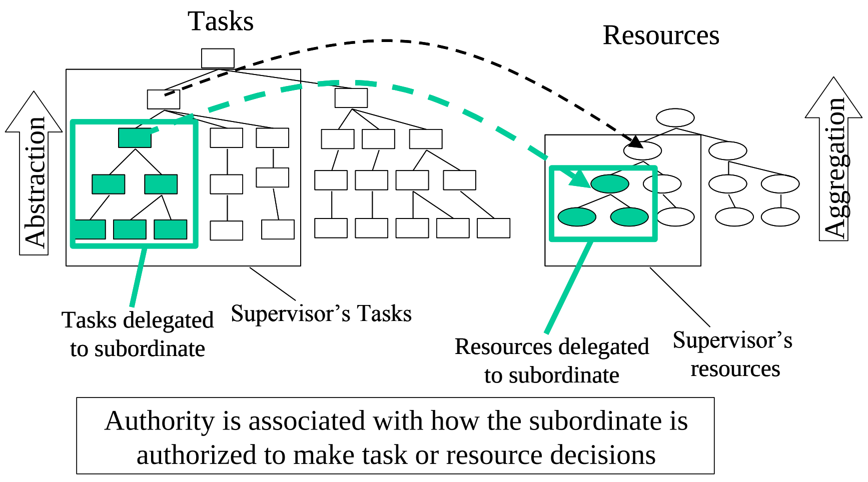

Tasks
Resources
Abstraction
Aggregation
Tasks delegated to subordinate
Supervisor's Tasks
Resources delegated to subordinate
Supervisor's resources
Authority is associated with how the subordinate is authorized to make task or resource decisions

Figure 8. Delegation characterized by abstraction, aggregation and authority dimensions.

The authority to make those decisions need not be complete. The supervisor can assert constraints on how the subordinate makes those decisions and/or can require the subordinate to perform various degrees of checking and request approval before the proceeding with a plan, but there must be *some* authority to make those decisions handed over if there is to be any benefit from delegation. Our LoA3

[1] The choice of the term "aggregation" here to refer to the part-whole dimension of objects in a system, as well as the reference to a means-ends dimension of functions and sub-functions as an "abstraction" dimension, are conscious references to Vicente (1999) and Rasmussen's (1984) framework for Cognitive Work Analysis, which use the terms in a similar fashion. Plus, they provide nicely alliterative LOA abbreviations. We do not otherwise claim to be using the Cognitive Work Analysis framework here.

framework thus sees a "Level of Automation" (or, perhaps more accurately, a human-automation relationship) as a combination of tasks delegated at some level of abstraction with some level of authority and resources delegated with some level of authority to be used to perform that (and perhaps other) task(s). The "level of automation" in a human-machine system increases if the level of abstraction, level of aggregation or level of authority (on either abstraction or aggregation) increases.

Using the Delegation Space Model as an Adaptive Automation Architecture

In preceding sections, we laid out a descriptive architecture for the space within which delegation interactions at various automation levels occur. In this section, we will outline three ways in which the framework can provide utility.

- Using the Framework as an Experimental Control Tool
- Enhancing Abstraction Flexibility (and Putting it Under Experimenter Control)
- Enabling Aggregation Flexibility (and Authority Over It)

The theoretical approach outlined above can be used to very precisely describe very specific human-automation relationships. Our proposal is to also use this framework to both guide experimentation to explore different regions of the delegation space and to serve as an experimenter's tool to allow flexible and precise control of the kind and level of automation an operator can access and use in an experimental trial with a Playbook interface. This means giving the experimenter the ability to alter aspects of the task abstraction, the control authority and the aggregation of resources which the human and automation can jointly control. In this section, we will illustrate how this can work within a Playbook framework, and the uses to which it can be put in experimentation.

First, simply by aggregating low level behaviors (like waypoint following) and providing the planning intelligence to assemble them into contextually-appropriate, higher level, more complex and abstracted behaviors, Playbook *gives* the user alternate levels of abstraction to command automation. In previous Playbook implementations, the operator could flexibly decide to command at various levels of abstraction. To enable experimentation exploring variations in control relationships along the abstraction dimension, we propose to create an experimenter's interface to provide the ability to lock the operator out of one or more of these levels of abstraction on a task-by-task basis. This would allow, for example, creating an experimental condition in which the operator only can only command Overfly plays (cf. Figure 9), or only Fly-to-Target or Fly-to-Waypoint subplays (i.e., command by inserting targets or individual waypoint sequences), or only Flight Control actions (via joystick controls), or has flexibility over only Overfly through waypoints (but not joystick commands), etc. This was accomplished in Phase 1, although the mechanism is unwieldy, and will need to be improved.

But if this were all we did, we would only afford experimenter control over the abstraction dimension. In Phase 2, we will do more—providing control over authority and aggregation dimensions as well. To accomplish this, and thereby provide a very rich framework for experiments on alternate delegation styles, we will need to add two experimenter capabilities to the current Playbook—the ability to alter the level(s) of authority and the level(s) of aggregation that the Playbook operator has access to in a given experiment or setting.

When one currently delegates a play or function to automation using Playbook, one is authorizing Playbook's planning component to develop a plan for achieving that function and then submit it to the operator for approval. If the operator approves, Playbook executes the plan. This interaction style represents an “Approval” level of authority for the delegation of tasks at the “Overfly” level of abstraction[2]. Essentially, previous Playbook implementations have only used this single, homogenous level of authority for all tasks they gave the user access to. These authority levels are fixed in the current system, but they need not be. Figure 9 shows a task decomposition that indicates some of the options that setting alternate authority levels within the abstraction dimension could afford. To accomplish an Overfly task/function, four subtasks must be accomplished. Currently, commanding at the Overfly level delegates the same level of authority to all subtasks (Approval), but this needn’t be the case. An experimenter could configure an experimental run in which the operator delegates Full authority to Achieve Airborne (e.g., the automation can make it’s own decisions and execute them, without further operator review or approval, about how to get an suitable aircraft airborne), Approval authority to Fly-to-Target, Monitor authority for Photograph Target, and Inform authority for Destage—or any other combination of tasks x authority levels that made sense, was feasible within the capabilities of the automation and was of interest to the experimenter.

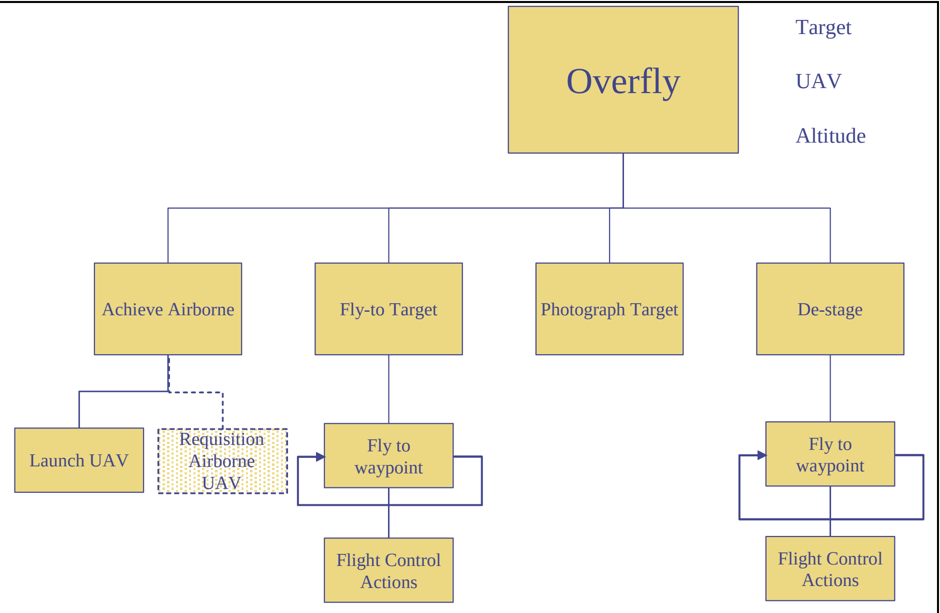


Figure 9. Overfly abstraction hierarchy

Although there is a tendency for delegated authority to increase at the lower levels of the task hierarchy, this need not be uniformly the case. For example, let’s say that there is a specific subsubtask, deep within the definition of Photograph Target (perhaps the use of an active imaging device like Radar that might compromise stealth). The automation could be given Full authority to plan and execute Overfly tasks—including all possible methods of accomplishing that parent task—*except* for this specific “Activate Radar” task. That task could be configured to have nothing higher than Approval authority (or even, perhaps, Monitor authority if it were an action the user had to take). In this case, Playbook would behave fully autonomously whenever it was asked to perform an Overfly play in all cases, *except* those where it deemed the use of the Activate Radar subsubtask desirable. In those cases, it would ask the user for approval to perform that specific task.

The above example illustrates the integration of Abstraction and Authority dimensions; a similar approach could work, together or separately, with the Aggregation dimension. Instead of associating a level of authority with a specific task (a specific part of the Abstraction dimension), it could work by

[2] In fact, we have altered this slightly in some Playbook implementations—primarily through giving Playbook to right to automatically begin execution of the plan it creates, while retaining the right to override it once it has begun (Override authority level).

asserting an authority level on a specific *resource* (the Radar system) and therefore be asserted against the Aggregation dimension.

Refining the LoA3 Architecture

Populating the LoA3 Matrix

Using the LoA^3 framework requires that specification of points or regions in the LoA3 space be distinguishable and repeatable. One to do this is establish values for each of the three dimensions of abstraction, authority, and aggregation, anchored by concrete examples. While this may seem to be an effort to quantify the scales, this need not be the case. The LoA-abstraction dimension is inherently qualitative rather than quantitative, although if a hierarchical task model is used, then a particular level along this dimension can be specified by a sub-task number within the hierarchy. LoA-authority is a semi-qualitative dimension that can be assigned numerical values. For example, automation for a flight task with levels of None, Monitor, Recommend, Override, Inform, and Full would be given values such as 0, 0.2, 0.4, 0.6, 0.8, and 1.0, respectively. Precise definitions of the flight task automation levels results in repeatable assignment of values across domains. Finally, LoA-aggregation is also a semi-qualitative dimension that can be assigned values (e.g., .25, .5 or .75 of all resources).

The various levels in the LoA^3 model can be combined to yield many possible human-automation schemes. Note, however, that the dimensions are not completely independent and that not all combinations are possible. For example, high LoA-abstraction—having automation fully responsible for carrying out a task, with no user involvement—would be incompatible with a value of "none" for LoA-authority or LoA-aggregation.

Populating the automation levels as "a function of mission complexity, information processing stages, and operator competence and training", previous approaches have used weighting factors that were used to determine appropriate LoAs. A related scheme was employed in another Army R&D program with which we were involved (Gacy, 2006). Briefly, weightings were developed for task importance, task connectedness, multitasking likelihood, expected workload, and the impact (on system performance) of automation. These weightings were combined with a very simple (though somewhat unique, see discussion below) one-dimensional LoA scale to formulate an "automation development priority", given by the formula:

Level of Automation x
Impact of Automation x
Multitasking likelihood x
Task connectedness x
Task Importance x
Expected workload x
100

For example, task importance was given a rating by a SME given the understanding that a task would be rated high on importance if it involved combat operations or contributed to unit survivability or engagement of the enemy. If so, then there would be a greater payoff of using automation for such a critical task, if available. Similarly, the expected workload associated with doing the task manually was estimated, with the idea that high-workload tasks would particularly benefit from use of automation. SME ratings of low, medium, or high on these and the other weighting factors were given values of 0.5, 0.8, or 1.0, respectively.

The LoA scale was given numerical values corresponding not only to the type of automation but also considering the *development cost* of the automation. This was apparently in response to a previous suggestion by Parasuraman et al. (2000), that the reliability and cost of a particular LoA should be taken into account by designers when considering automation in a particular system. For example, a highly reliable, nearly-fully automated system could be developed, but the cost (in terms of hardware and software engineering effort) might be so prohibitive as to make such a design option impractical. Wilson and Neal (2001) also examined the tradeoff between system performance and the cost of developing automation algorithms for human telerobotic control of a simulated “sheepdog” herding a group of simulated “sheep.” They showed that as a higher LoA was developed and applied to this system, there were diminishing returns in terms of system performance and a steep rise in programming cost. Reflecting these and other related considerations, Gacy (2006) used the following LoA scale values: No automation (manual performance) was given a value of 0.1, so that values could be compared among all tasks with no automation, if desired. Automation that provided decision aiding but kept the operator in the action response loop was given a value of 1, perhaps reflecting a value judgment (supported by some empirical research; see Parasuraman et al., 2000; Rovira et al., 2007) that this is an optimal form of automation. Partial automation, in which the task was shared between human and automation, was rated 0.7. Finally, full automation was rated 0.5, with the view that such automation would have the poorest payoff to development ratio.

Combining the LoA value with the weightings yields the automation development priority index. The higher this value, the greater the benefit of using a particular LoA for a task. For example, for a decision aid automation tool that had medium impact for a high-importance task with low multi-tasking likelihood, medium task connectedness, and high workload, the index would be 100 x 1 x .8 x 1 x .5 x .8 x 1 = 32.

Using this approach, Gacy (2006) determined a “top task” for automation (highest automation development index = 51) for a FCS mounted platoon:

Task: Determine Movement Technique

2 The Maneuver Battlefield Operating System
2.2: Conduct Tactical Maneuver
2.2.1 Employ Combat formations
2.2.1.1 Employ Traveling Movement Technique

Determine Movement Technique
Automation Development Priority = 51

The recommended LoA for this task was a decision aid, in the form of a route planner, combined with orders to determine best what type of traveling movement technique was needed, such as Traveling, Traveling Overwatch, or Bounding Overwatch.

While the reduction to a single value is appealing, the simplicity of the approach leads to some hidden assumptions and seemingly awkward formulations (e.g., the implicit value judgment about operator engagement in the task).

Instead of adopting this model, we have considered the factors that would affect the choice of a particular region in LoA3 space, and realized that our formulation had a 'missing link'.

Missing Links

It has become apparent as we have worked through Phase 1 that the LoA3 model has a serious omission if it is to be used (as we proposed) as the framework to develop an experimental testbed and use it to conduct experiments into human-automation delegation relationships.

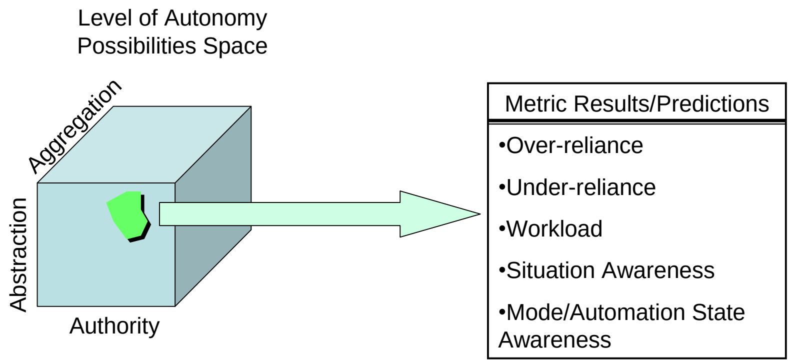

Level of Autonomy Possibilities Space
Aggregation
Abstraction
Authority
Metric Results/Predictions
•Over-reliance
•Under-reliance
•Workload
•Situation Awareness
•Mode/Automation State Awareness

Figure 10. Phase 1 formulation

Figure 10 shows the formulation of the LoA3/Playbook testbed we proposed in Phase 1. A point or region in LoA3 space is defined, and estimated in simulation. This gives rise, either in informal demonstration, or in more formal experimentation, to several metrics that can indicate how appropriate that particular region would be as a selected LoA3. This begs the question, "Appropriate in the face of what conditions?" One of our advancements in Phase 1 of this work has been to flesh out an answer to that question and to develop a plan for incorporating it into our plans for Phase 2.

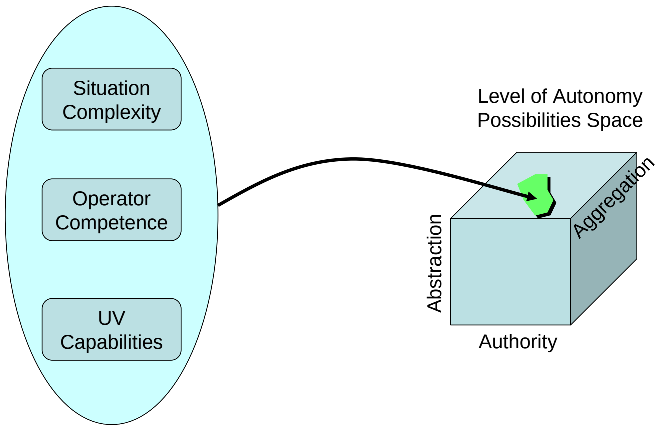


Figure 11. Predictive mapping

Figure 11 shows the missing piece – there needs to be some explicit knowledge of, and repeatable representation of the environment in which you are operating to be able to predict what the appropriate region of LoA^3 space would be.

Mathematically, one could view this as a mapping $f:\Re^3 \rightarrow \Re^3$, or

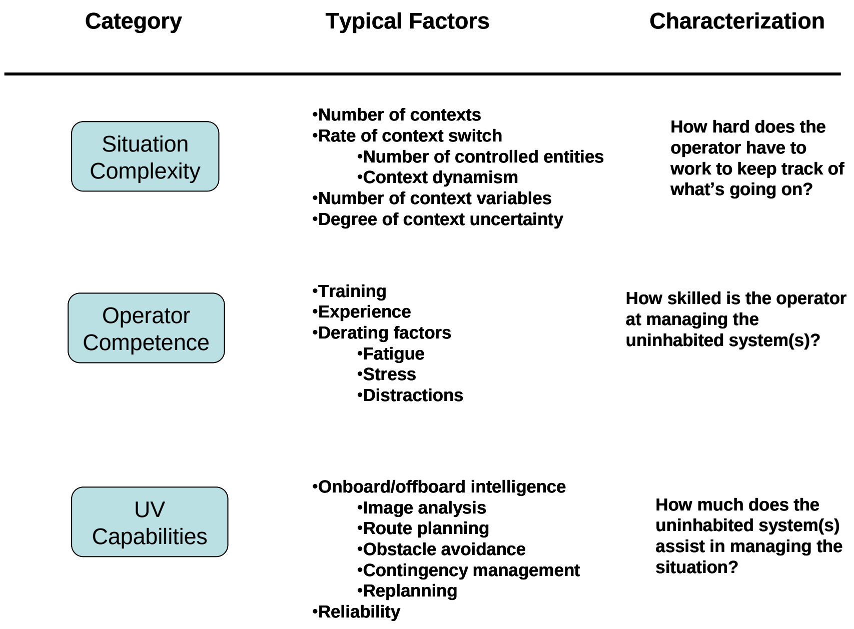
$$(Complexity, Competence_{Op}, Capabilities_{UV}) \rightarrow (Authority, Abstraction, Aggregation).$$

While the function is *total* – that is, every value of Complexity X Capabilities$_{Op}$ X Capabilities$_{UV}$, (or C^3 space) has an 'answer' or value in LoA^3 space, the function is likely neither *one-to-one* nor *onto* – that is, there could be several values of Authority X Abstraction X Aggregation which are equally optimal answers in terms of the metric values which result, and there are likely values in the LoA^3 space which are appropriate for no point in C^3 space.

Figure 12 shows several factors that we will consider as elements to characterize the ‘input space’ for such a predictive function. In Phase II, we will more carefully elaborate this set of factors and a means to repeatably denote a particular input combination.

The CLAMP^3 model

Together, the input factors of Complexity of the Situation, Competence of the Operator and Capabilities of the Automation (C^3); plus the operating range parameters of Level of Authority, Level of Abstraction and Level of Aggregation ((LA)^3) as a Mapping for Prediction of Personnel Performance give the CLAMP^3 model. The reader may note that there are not three ‘M’s in the acronym, making it somewhat flawed. Nonetheless we like the association with the idea of getting a firm grip on the necessary factors involved in making the correct choices.



Category	Typical Factors	Characterization
Situation Complexity	•Number of contexts •Rate of context switch •Number of controlled entities •Context dynamism •Number of context variables •Degree of context uncertainty	How hard does the operator have to work to keep track of what’s going on?
Operator Competence	•Training •Experience •Derating factors •Fatigue •Stress •Distractions	How skilled is the operator at managing the uninhabited system(s)?
UV Capabilities	•Onboard/offboard intelligence •Image analysis •Route planning •Obstacle avoidance •Contingency management •Replanning •Reliability	How much does the uninhabited system(s) assist in managing the situation?

Figure 12. Typical factors for LoA^3 selection

Modify Playbook to Support Demonstration

Defining Playbook Modifications for Use as an LoA3 Testbed

We have determined that the following modifications to previous Playbook implementations will be necessary in order to use it as a testbed for exploring the LoA3 space of delegation alternatives:

Enhancing Abstraction Flexibility (and Putting it Under Experimenter Control)

To make this framework a reality, we intend to implement portions of the delegation framework as follows. (Note that implementation of some of these capabilities has begun in Phase 1, while others are reserved for Phase 2, as will be described in subsequent sections.) First, we will provide a UI, structured around a task tree expansion of the underlying Playbook Task Model, to enable an *experimenter* to selectively “turn on and off” levels of the abstraction hierarchy. This allows experimenters to allow operators to command Overfly or waypoint commands only, or the intermediate level subplays. This interface allows a range of flexibility to command across the abstraction levels (Overfly plays and intermediate subplays but not waypoint commands, etc.)

Enabling Authority Flexibility over Tasks (Authority x Abstraction interactions)

Next, we have decided to add some alternate (and experimenter selectable) levels of authority to these abstraction levels. Providing multiple authority levels at the Overfly abstraction level should be comparatively straightforward.

We provide the experimenter the ability to configure which of these authority levels is available for which tasks for the operator to delegate to automation for each experimental run—for example, allowing the operator Approval authority only for “Photograph Target” in one run, and only Monitor authority in the next. Allowing the experimenter the ability to provide the operator flexible control across authority levels is more difficult, since it entails determining which levels of authority are supportable in Playbook connected to various simulation environments, but is clearly desirable.

Enabling Aggregation Flexibility (and Authority Over It)

Aggregation works analogously to Abstraction. An early step will be to enable the assertion of variable authority levels (by the experimenter) on the use of specific UAVs. For example, the APC can currently use any UAV it desires (that has not already been tasked) with Override authority, but we could configure this so that Full, Inform and Approval authority levels are settable—with regards to all or to any specific UAV. Recommend authority could be achieved by allowing the APC to recommend UAV usage, and Monitor authority would allow the APC to critique the operator’s choice of UAVs. We could provide the ability to set these authorization levels, against all or against specific UAVs, as a part of the experimenter’s UI.

To the extent that there are existing groupings of resources, automation can be flexibly granted Authority for any group. To the extent that it is interesting, new groupings can be defined, analogous to the insertion of new levels of abstraction in the task hierarchy.

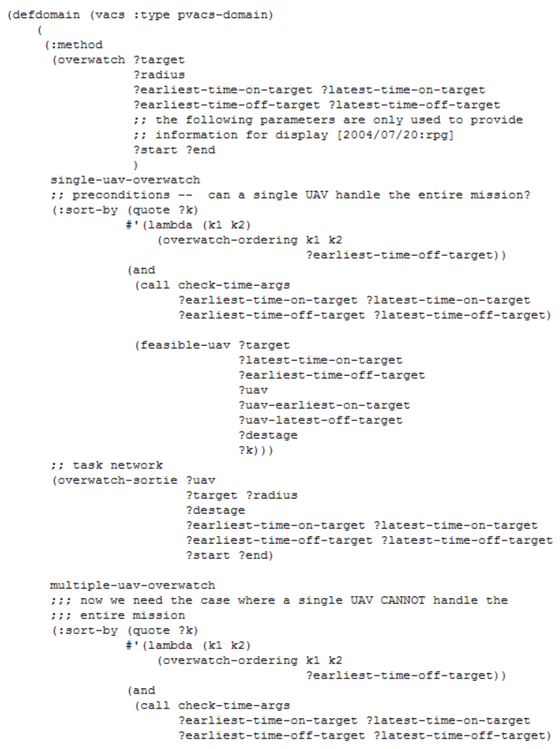
Automation Alternatives in Overfly Example

In using this architecture with a comparatively simple Overfly play, we have defined at least three levels of Abstraction (Overfly Play, Intermediate Subplays, and Waypoint-level Sub-subplays) and seven levels of authority on the abstraction levels. We have also identified at least three levels of Aggregation (UAV groups or teams, individual UAVs and UAV subsystems) crossed by the same seven levels of authority. This provides a total of 3x7x3x7 (or 441) different human-automation interaction styles within the testbed. While not all of these levels may be technically feasible or worth investigating, this illustrates how the proposed Playbook-based LoA3 enables the systematic investigation of a huge range of alternate human-automation interaction styles.

Using the Framework to Configure Human-Automation Roles and Responsibilities

The example above assumes that the experimenter has the control to turn on and off the abstraction, aggregation and authority levels for the operator either in an a priori, pre-mission setting or dynamically during mission execution. There is however, great utility in changing that arrangement to give similar control to the operator. To do this, we would provide an interface for the operator to configure the permissions with which automation may operate when asked to plan and/or execute a play. This would be an action that the operator (or engineering support staff) would perform pre-mission, typically during non-deployment down time, or at least prior to tasking a semi-automated asset(s). It would represent a significant enhancement to the Playbook architecture we have developed to date and, we suspect, would substantially improve its commercialization potential. There are a great many potential customers in the world who need to be able to interact with automation in a way that is both easy and yet predictable; more than there are who need to perform a wide range of experiments on human-automation interaction.

Playbook Modification Implementation

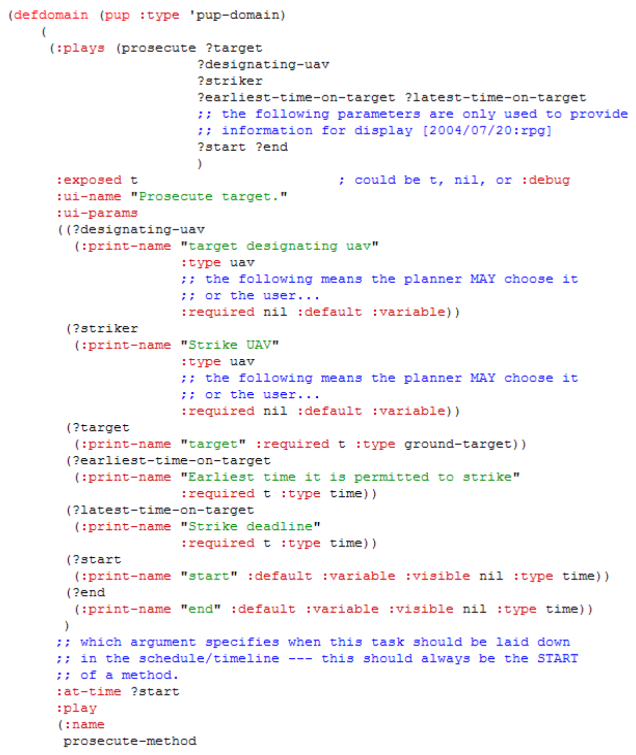


```
(defdomain (vacs :type pvacs-domain)
    (
     (:method
      (overwatch ?target
                 ?radius
                 ?earliest-time-on-target ?latest-time-on-target
                 ?earliest-time-off-target ?latest-time-off-target
                 ;; the following parameters are only used to provide
                 ;; information for display [2004/07/20:rpg]
                 ?start ?end
                 )
      single-uav-overwatch
      ;; preconditions --  can a single UAV handle the entire mission?
      (:sort-by (quote ?k)
                #'(lambda (k1 k2)
                    (overwatch-ordering k1 k2
                                        ?earliest-time-off-target))
                (and
                 (call check-time-args
                       ?earliest-time-on-target ?latest-time-on-target
                       ?earliest-time-off-target ?latest-time-off-target)

                 (feasible-uav ?target
                               ?latest-time-on-target
                               ?earliest-time-off-target
                               ?uav
                               ?uav-earliest-on-target
                               ?uav-latest-off-target
                               ?destage
                               ?k)))
      ;; task network
      (overwatch-sortie ?uav
                        ?target ?radius
                        ?destage
                        ?earliest-time-on-target ?latest-time-on-target
                        ?earliest-time-off-target ?latest-time-off-target
                        ?start ?end)

      multiple-uav-overwatch
      ;;; now we need the case where a single UAV CANNOT handle the
      ;;; entire mission
      (:sort-by (quote ?k)
                #'(lambda (k1 k2)
                    (overwatch-ordering k1 k2
                                        ?earliest-time-off-target))
                (and
                 (call check-time-args
                       ?earliest-time-on-target ?latest-time-on-target
                       ?earliest-time-off-target ?latest-time-off-target)
```

Figure 13. Previous Playbook representation



```
(defdomain (pup :type 'pup-domain)
    (
     (:plays (prosecute ?target
                        ?designating-uav
                        ?striker
                        ?earliest-time-on-target ?latest-time-on-target
                        ;; the following parameters are only used to provide
                        ;; information for display [2004/07/20:rpg]
                        ?start ?end
                        )
      :exposed t                                ; could be t, nil, or :debug
      :ui-name "Prosecute target."
      :ui-params
      ((?designating-uav
        (:print-name "target designating uav"
                     :type uav
                     ;; the following means the planner MAY choose it
                     ;; or the user...
                     :required nil :default :variable))
       (?striker
        (:print-name "Strike UAV"
                     :type uav
                     ;; the following means the planner MAY choose it
                     ;; or the user...
                     :required nil :default :variable))
       (?target
        (:print-name "target" :required t :type ground-target))
       (?earliest-time-on-target
        (:print-name "Earliest time it is permitted to strike"
                     :required t :type time))
       (?latest-time-on-target
        (:print-name "Strike deadline"
                     :required t :type time))
       (?start
        (:print-name "start" :default :variable :visible nil :type time))
       (?end
        (:print-name "end" :default :variable :visible nil :type time))
       )
      ;; which argument specifies when this task should be laid down
      ;; in the schedule/timeline --- this should always be the START
      ;; of a method.
      :at-time ?start
      :play
      (:name
       prosecute-method
```

Figure 14. Enhanced Playbook representation

While we have performed definition of multiple plays in multiple domains (see **Error! Reference source not found.** below), the internal representation used for play definition, shown in Figure 13 reflects the input format of the planning engine, the Simple Hierarchical Ordered Planner (SHOP2). It is not intuitive, and would not be suitable for manipulation by experimenters, who will need to modify this play structure (in some fashion) to implement constraints on authority, abstraction and aggregation selection. Indeed, even within SIFT, coming back to revise a previous play definition is an error prone proposition, since there are multiple positional or subtle form differences that are meaningful to SHOP2. In the Phase 1 effort, we have made a step toward improving this situation by adding explicit tags to portions of the Playbook form, shown in Figure 14, which are then translated by another module into the SHOP2 input format.

This new format has the added advantage of combining several aspects of the definition which had been split across four files into a single location. Still, this form requires a learning curve, and we have begun work to define a graphical authoring environment for simple play modifications. This will be more suitable for experimenter use in Phase II, since it will require a much smaller learning curve. More sophisticated changes would still, of course, require interaction with the complete textual representation.

In addition, we have been implementing a node selection interface to allow specification of internals for an arbitrary step in the plan. This is not complete, though we have remaining time and funds in Phase 1 to make further progress.

Demo/Eval LoA3 Testbed

A first step in the demonstration/evaluation was to establish what we were interested in showing, albeit informally in Phase 1, and how we might go about showing those effects. This led to the identification and elaboration of a number metrics that will be used in Phase II, some of which can be shown in the Phase 1 demonstration.

Resolution Metrics

The ability to vary the delegation approach across the LoA^3 dimensions is only of passing interest if we are not able to assess the effects of various approaches on the known issues in human-automation interaction. In the following section, we identify metrics which will be used to directly address – and resolve – those issues in our Phase 2 work. We will term these "Resolution Metrics". Resolution metrics are used to directly measure the results of selecting a particular region within the LoA^3 space.

As Kaber and Endsley (2004) pointed out, determining the most adequate human-automation architecture is not a simple endeavor. Several factors need to be taken into account, including over- and under-reliance, out-of-the-loop performance (OOTLP), trust, workload, situation awareness, mode awareness and knowledge of automation state, and ultimately, human-automation performance. To be able to examine these topics effectively and determine the most appropriate human-automation architecture, it is necessary to consider a set of resolution metrics that can address each of these issues.

Over Reliance and Out-Of-The-Loop Performance (OOTLP)

Out of the loop performance (OOTLP) occurs when human operators are forced to manually intervene with the functioning of a system after a period of performing with the aid of an automated system. Prior research indicates that there are several problems associated with OOTLP. In general, human operators have difficulties in both detecting system malfunctions during automated control and regaining effective manual control (Wiener & Curry, 1980).

There are different reasons for these two types of OOTLP performance decrements. Humans have a tendency to overtrust automated systems, particularly when the automated systems are highly reliable (Lee & See, 2004). Consequently, operators tend to be overly reliant on automated systems and fail to monitor their performance effectively, a phenomenon referred to as complacency (Parasuraman & Riley, 1997). Also, prior research suggests that operators require additional time and resources to reorient themselves after regaining manual control from automated control (Wickens & Kessel, 1981). This, in turn, limits their ability to actually perform the tasks that they are required to perform (Kaber & Endsley, 2004).

In addition to these two factors, it is important to consider the notion that the type and amount of feedback that the automated system offers human operators is a contributing factor to OOTLP decrements (Endsley & Kiris, 1995). Different levels of automation support different amounts of feedback. For example, a full level of automation offers no direct feedback to the operator as far as its state or functioning. In contrast, lower levels of automation provide a greater amount of direct feedback to operators either by simply informing them of the current functions of the system or by requesting their permission before carrying out actions. Therefore, the level of automation used may ultimately serve as a contributing or a mitigating factor to OOTLP decrements.

There are two potential ways in which to examine over reliance and OOTLP. One method consists of introducing random automation malfunctions and examining the accuracy and latency with which human operators can both detect such malfunctions and make corrective actions by resuming manual control of the task. The other approach is to require operators to alternate between manual and automated control using a dynamic function allocation architecture. Prior research suggests that both of these approaches are effective ways of assessing over reliance and OOTLP (Endsley & Kaber, 1999; Endsley & Kiris, 1995; Kaber & Endsley, 2004).

Under Reliance

Under reliance can also be problematic to the extent that human operators' fail to take full advantage of the automation's capabilities. One of the main reasons why humans may under rely on automation is lack of trust (Lee & See, 2004). Trust serves as a mediating factor between humans and automated systems. It is a common error (cf. UAV Roadmap 2005) to say that it is desirable to *maximize* trust in automation. This would result in the over-reliance situation above. Rather, it is desirable to achieve an accurate belief about the automated system's capabilities, or "appropriate trust".

There are at least two ways in which to effectively examine the issue of under reliance. One way of accomplishing this is to examine operators' trust on the automated system. Trust in automation is typically defined as the attitude that a given automated system will aid in accomplishing an operators' goal under conditions of uncertainty (Lee & See, 2004). As such, a typical method of measuring trust is through the use of self-report measures. For the purpose of this SBIR, we could either use single-indicator measures of trust as previously used by other researchers (Muir & Moray, 1996) or a multiple-indicator scale specifically developed for this purpose (Jian, Bisantz, & Drury, 2000).

Another method of more directly assessing the issue of under reliance is to quantify the proportion of time that operators perform each task with the aid of the automated system using a dynamic function allocation architecture. Furthermore, we could even generate a more sensitive automation reliance metric by taking into account not only the proportion of time of automation usage but also the level of automation selected by the operator.

Workload

The primary reason for automating tasks is to reduce operators' workload, thereby allowing them to focus more of their resources on higher level strategic planning. However, automation does not always lead to lower workload (Parasuraman & Riley, 1997). Although automation may reduce the workload associated with performing certain functions, it may also increase operators' workload by imposing additional monitoring demands, or by becoming unreliable at specifically those difficult times that the operator most needs assistance. Consequently, it is essential to examine the level of workload created by a given human-automation architecture to ensure that the task demands do not exceed operators' capabilities, not only under nominal conditions, but more importantly under high complexity and uncertainty conditions.

There are several ways of assessing operators' workload, but the most frequently used are secondary-task performance and self-report measures (Wierwille & Eggemeier, 1993). Secondary task performance measures are based on the notion that humans have a certain amount of resources, which they can allocate to different tasks. The idea is that given a primary task, humans can still perform a secondary task provided that they have spare mental capacity. As workload on the primary task increases, however, performance on the secondary task deteriorates, thereby serving as an indicator of the amount of spare mental resources. A potential disadvantage of using this approach is that the secondary task can be intrusive, and it may also create an unrealistic working environment (Williges

& Wierwille, 1979). A possible way of overcoming this limitation is to use an embedded secondary task that is part of the overall working environment. In the case of monitoring and controlling unmanned aerial vehicles (UAVs), a candidate embedded secondary task could be a communication task facilitated through the use of a chat box. Prior research suggests that this could be an effective method for measuring workload in such a task environment (Cummings & Guerlain, 2004).

Another method of measuring workload is using self-report measures, such as the NASA TLX (Hart & Staveland, 1988), the Subjective Workload Assessment Technique (Reid & Nygren, 1988), and the Workload Profile (Tsang & Velazquez, 1996). The main advantages of using these types of measures are that they are easy to administer, they have high face validity, and they are relative unobtrusive (Wierwille & Eggemeier, 1993). Potential issues of concern associated with these measures include reliability and measurement invariance (Bustamante, Bailey, Fallon, Newlin, & Bliss, 2005). Nevertheless, using self-report measures in conjunction with other workload assessment methods, such as secondary-task performance, can be a powerful way of assessing operators' workload (Matthews, Davies, Westerman, & Stammers, 2000).

Situation Awareness

Maintaining a high level of situation awareness is essential for effective performance of complex tasks, especially when operators interact with automated systems (Endsley, 1996). One of the most effective methods of assessing operators' situation awareness is the Situation Awareness Global Assessment Technique (SAGAT; Endsley, 1995). Similarly to the assessment of workload, in addition to the SAGAT, it may also be beneficial to complement the assessment of situation awareness with a self-report measure, such as the Situation Awareness Rating Technique (SART, Taylor, 1988).

Mode Awareness and Knowledge of Automation State

Prior research suggests that mode awareness and knowledge of automation state are essential components of effective human-automation performance (Sarter & Woods, 1995). In essence, both of these issues are intrinsically subsumed under the broader umbrella of situation awareness. As such, they can be effectively measured using a similar approach to that of measuring situation awareness. However, it is useful to separate situation awareness into categories of internal situation (vehicle health, automation mode, etc.) and external situation (opposing forces' stance, environmental factors, etc.). Situation Awareness is sometimes used to refer only to this external state. We will use a modified SAGAT approach designed specifically to target aspects of internal state concerning mode awareness and knowledge of automation state.

Current Demonstration Status

As the Phase 1 effort is not yet complete (approximately 20% of time and funds remaining), the demonstration has not been completed. We have obtained permission, and the requisite hardware/software to install MUSE at Dr. Parasuraman's lab. We have performed informal measurements, principally of learning curve and setup time, to see the effects of the different LoA3 spaces for the two demonstration cases. The differences for the learning curve and setup times are substantial. We anticipate that demonstrations at NASA Ames and CSL will provide clear indications of other differences as well.

Conclusion

We have presented a description of a testbed approach for examining various approaches to delegation as a means of supervisory control. The testbed builds on existing Playbook software, using a three-dimensional delegation space, the LoA^3, to characterize the level of automation provided by the current Playbook testbed configuration. We anticipate that any given context, characterized in terms of (1) situation complexity, (2) operator capabilities and (3) automation capabilities, can be mapped to one or more regions of LoA^3 space which will result in optimized performance of operator+automation for that context.

References

Billings, C. (1997). *Aviation Automation: The search for a human-centered approach*. Mahwah, NJ: Lawrence Erlbaum.

Bone, E. and Bolkcom, C. (2003). *Unmanned Aerial Vehicles: Background and Issues for Congress*. Congressional Research Service Technical Report #RL31872.

Cable News Network. (2002). Nearly half Air Force's UAV Predators lost since deployment. Retrieved January 16, 2003 from the World Wide Web: www.rense.com/general33/half.html.

Degani, A. (2004). *Taming Hal: Designing interfaces beyond 2001*. New York: Palgrave.

Gacy, M. (2006). *HRI ATO automation matrix: Results and description*. Technical report, MicroAnalysis and Design, Inc., Boulder, CO.

Jones, P. & C. Jasek (1997). "Intelligent Support for Activity Management (ISAM): An architecture to support distributed supervisory control*," IEEE Trans. on Sys. Man & Cyber.—Pt. A: Sys & Hum, vol. 27*, pp. 274-288.

Kirlik, A. (1993). Modeling strategic behavior in human-automation interaction: Why an 'aid' can (and should) go unused. Human Factors, 35, 221-242.

Lee, J. D. & See, K. A. (2004). Trust in computer technology: Designing for appropriate reliance. *Human Factors, 46*(1), 50-80.

Manning, S., Rash, C., LeDuc, P., Noback, R. & McKeon, J. (2004). *The Role of Human Causal Factors in US Army UAV Accidents*. US Army Aeromedical Research Laboratory Technical Report #2004-11.

Miller, C. "Delegation architectures: Playbooks & policy for keeping operators in charge," in *Proceedings of the NATO Research and Technology Organization Special Workshop on Uninhabited Military Vehicles*, (Leiden, Holland), June 2003.

Miller, C., Funk, H., Goldman, R., and Wu, P. (2003). 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*, March 22-25, Dayton Beach, FL.

Miller, C. and Goldman, R. (1997). "Tasking Interfaces: Associate Systems that know Who's the Boss." In *Proceedings of the 4th International Workshop on Human-Computer Teamwork*. Krueth, Germany; September 23-27.

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. and Parasuraman, R. (2007). Designing For Flexible Interaction Between Humans and Automation: Delegation Interfaces for Superviory Control. *Human Factors*, Vol. 49, No.1, February 2007, pp. 57-75.

Miller, C. and R. Parasuraman (2003). "Beyond levels of automation: An architecture for more flexible human-automation collaboration," in *Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society*, (Denver, CO), Oct. 2003.

Nau, D., et al., Applications of SHOP and SHOP2. CS-TR-4604, UMIACS-TR-2004-46 http://www.cs.umd.edu/~nau/papers/nau04applications.pdf.

Pande, A. & C. Hayes. (2002). "Plan-Edit: An Interactive Critic for User Guided Generation of High Quality Manufacturing Setup Sequences", In *Proc. of ASME Int'l Mtg. Cong. & Exposition*, Montreal, Canada.

Parasuraman, R., Galster, S., Squire, P., Furukawa, H., & Miller, C. (2005). A flexible delegation-type interface enhances system performance in human supervision of multiple robots: Empirical studies with RoboFlag. *IEEE Transactions on Systems, Man, and Cybernetics. Part A. Systems and Humans, 35*, 481-493.

Parasuraman, R., & Miller, C. (2006). Delegation interfaces for human supervision of multiple unmanned vehicles. In N. Cooke, H. L. Pringle, H. K. Pedersen, & O. Connor (Eds). *Human Factors of Remotely Operated Vehicles. Advances in Human Performance and Cognitive Engineering*. Volume 7, Oxford, UK: Elsevier.

Parasuraman, R. & Riley, V. (1997). Humans and automation: Use, misuse, disuse, abuse. *Human Factors, 39*, 230-253.

Parasuraman, R., Sheridan, T.B., Wickens, C.D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, *30*, 286-297.

Rasmussen, J., Pejtersen, A. and Goodstein, L. (1994). *Cognitive Systems Engineering*. New York; Wiley.

Ruff, H. A., Narayanan, S. and Draper, M. H. "Exploring Automation Issues in Supervisory Control of Multiple UAVs," *Proceedings of the Human Performance, Situation Awareness, and Automation Technology Conference*, March, 2004, pp. 218-222.

Schneiderman, B. (1993). *Designing the User Interface*. Reading, MA: Addison-Wesley.

Shappell, S.A. and Wiegmann, D.A. (2000). Human factors analysis and classification system--HFACS. Department of Transportation, Federal Administration: Washington, DC. DOT/FAA/AM-00/7.

Sheridan, T. (1987). Supervisory Control. In G. Salvendy, (Ed.) *Handbook of Human Factors.* New York: John Wiley & Sons. 1244-1268.

Sheridan, T. and W. Verplank, (1978). "Human and computer control of underea teleoperators," Technical Report, MIT Man-Machine Systems Laboratory, Cambridge, MA, 1978.

Sheridan, T., & Parasuraman, R. (2006). Human-automation interaction. *Reviews of Human Factors and Ergonomics, 1,* 89-129.

Squire, P., Trafton, G., & Parasuraman, R. (2006). Human control of multiple unmanned vehicles: Effects of interface type on execution and task switching times. In *Proceedings of the First Human-Robot Interaction Conference,* Salt Lake City, UT.

Vicente, K. (1999). *Cognitive work analysis*. Mahwah, NJ; Erlbaum.