Submitted for presentation at the International Conference on Intelligent User Interfaces. Redondo Beach, CA: January 5-8, 1999.

"Tasking" Interfaces for Flexible Interaction with Automation: Keeping the Operator in Control

Christopher A. Miller, Michael Pelican and Robert Goldman
Honeywell Technology Center
3660 Technology Dr.
Minneapolis, MN 55418 U.S.A.
cmiller, pelican, goldman@htc.honeywell.com

ABSTRACT

The ongoing debate in the HCI community between direct manipulation and intelligent, automated agents points to a fundamental problem in complex systems. Humans want to remain in charge even if they don't want to (or can't) make every action and decision themselves. We have been exploring a middle road through the development of "tasking interfaces"—interfaces which share a task model with a projective planning system to enable human operators to flexibly "call plays" (that is, stipulate plans) at various levels of abstraction, leaving the remainder of the plan to be fleshed out by the planning system. The result is akin to 'tasking' a knowledgeable subordinate to whom one can give more or less detailed instructions. We describe a prototype tasking interface for developing mission plans for Uninhabited Combat Air Vehicles (UCAVs).

Keywords

Tasking Interface, Mixed Initiative Planning and Control, Task Model, Hierarchical Task Network Planning, Uninhabited Combat Air Vehicles

INTRODUCTION

As systems become more complex, there is increasing temptation to control them via "automation" [e.g., 1]—either in the form of subsystems which fully perform the task, or as 'decision aids' which provide guidance but leave the final execution to the human. In spite of extensive technological achievements in automation technology (as well as some spectacular failures), experience has consistently shown that advanced automation suffers from a basic sociological problem. Human operators of complex systems want to remain in charge. For example, in developing the Rotorcraft Pilot's Associate Cockpit Information Manager [10], we multiple pilots and designers to develop a consensus list of prioritized goals for a "good" cockpit configuration manager. Two of the top three items on the list were "Pilot remains in charge of task allocation" and "Pilot remains in charge of information presented."

There are good reasons to design and use systems at the higher levels of automation. By definition [17], such systems share responsibility, authority and autonomy over many work behaviors with human operator(s) to accomplish their goals of reducing operator workload and information overload. While operators may wish to remain in charge, and it is critical that they do so, today's complex systems no longer permit them to be fully in charge of all system operations—at least not in the same way as in earlier cockpits and workstations.

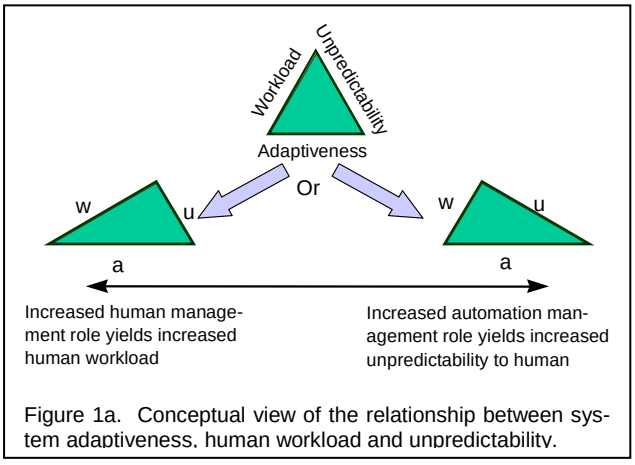


Figure 1a. Conceptual view of the relationship between system adaptiveness, human workload and unpredictability.

Conceptually, the problem can be presented as in Figure 1a. This figure shows the relationship between the adaptiveness of a human-machine system as a function of the workload or unpredictability it causes for the human operator. This view implies that for any increase in adaptiveness (the ability of the human-machine system to perform in an appropriate, context-dependent manner across situations) there must be an accompanying increase in one or both of the other two legs of the triangle. Either human workload (the amount of physical, attentional or cognitive "energy" the human must exert to use the system) or unpredictability (inability of the human to know what the automation will do at any given time) must increase.

Since adaptiveness is generally the goal of added complexity (though systems can be complex without achieving it), this is equivalent to saying that any increase in human-machine system complexity must affect the human operator in two ways—either (1) the added complexity must be fully controlled by the human, resulting in increases in workload,

or (2) the added complexity must be managed by automation, resulting in increases in unpredictability. The ongoing debate [9,18] in the HCI design community over intelligent agents vs. direct manipulation interfaces is a manifestation of these alternative approaches.

In recent work developing an interface for Uninhabited Combat Air Vehicles (UCAVs), we have explored a middle road between these alternatives. In brief, our solution is to allow human operators to interact with advanced automation *flexibly* at a variety of levels. This allows the human operator to smoothly vary the 'amount' of automation s/he uses depending on such variables as time available, workload, criticality of the decision, degree of trust, etc.—variables known to influence human willingness and accuracy in automation use [13]. It further allows the human to flexibly act within the limitations imposed by the capabilities and constraints of the equipment and the world—a strategy shown to produce superior aviation plans and superior human understanding of plan considerations in [8] and which lies at the heart of recent advances in ecological interface design [19]. While this does not eliminate the dilemma presented in Figure 1a, it mitigates it by allowing operators to choose various points on the spectrum for interaction with automation (see Figure 1b). We call human-machine systems of this sort "tasking" interfaces, because they allow posing a task to automation at all the different levels one might 'task' a knowledgeable subordinate.

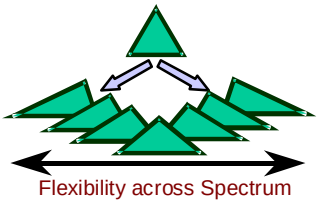


Figure 1b. A 'Tasking Interface' provides multiple methods of increasing adaptivenss.

PROPOSED SOLUTION— TASKING INTERFACES

There are three primary challenges involved in the construction of a tasking interface:

(1) A shared vocabulary must be developed, through which the operator can flexibly pose tasks to the automation and the automation can report how it intends to perform those tasks.

(2) Sufficient knowledge must be built into the interface to enable making intelligent choices within the tasking constraints imposed by the user.

(3) One or more interfaces must be developed which will permit inspection and manipulation of the tasking vocabulary to pose tasks and review task elaborations in a rapid and easy fashion.

Figure 2 presents our general architecture for tasking interfaces. The three primary components each address one of the challenges described above. In our approach, a Graphical User Interface (GUI) in the form of a "playbook" and a Mission Analysis Component (MAC) are based on and communicate with each other and with the human operator via a Shared Task Model. The human operator communicates tasking instructions in the form of desired goals, tasks, partial plans or constraints, via the Playbook GUI, in accordance with the task structures defined in the shared task model. These are, in fact, the methods used to communicate commander's intent in current training approaches for U.S. battalion level commanders [16]. The MAC is a *projective* planning system capable of understanding these instructions and (a) evaluating them for feasibility or b) expanding them to produce fully executable plans. The MAC may draw on special purpose planning tools to perform these funcions, wrapping them in the task-sensitive environment of the tasking interface as a whole. Outside of the tasking interface itself, but essential to its use, are two additional components. Once an acceptable plan is created, it is passed to an Event Handling component, which is a *reactive* planning system capable of making moment by moment adjustments to the plan during execution. The Event Handling component then passes these instructions to control algorithms that actually effect behaviors in the controlled system automation. Since the GUI, MAC and the Shared Task Model are components unique to the construction of a tasking interface, they will be described in more detail in separate subsections below.

Shared Task Model

A critical enabling technology for tasking interfaces is the ability to code, track and dynamically modify the goals and plans which human users have in operating their automation and equipment. By explicitly representing these entities in a "task model" format that is *both* familiar to a human operator *and* interpretable by a knowledge-based planning system, we gain a level of coordination between human and system beyond that previously possible.

Beyond the comparatively static task models used in workflow support tools [14], a task model must meet several requirements to support a tasking interface. First, it must be organized via functional decomposition—each task or goal must be decomposable into alternate methods of achieving that goal, down to some bottom layer of executable, primitive actions. A 'task', therefore, represents an encapsulated set of behaviors—perhaps including alternatives—which is known as a method of accomplishing domain goals. As Boy [2] points out, the naming or labeling of tasks always in-

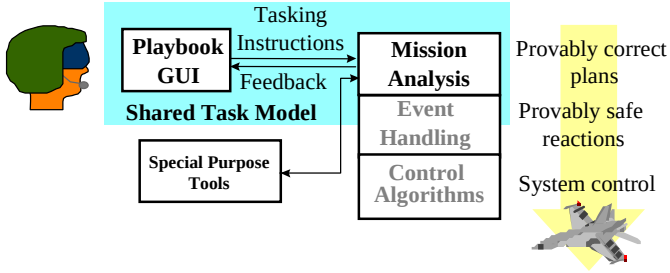


Figure 2. General Architecture for Tasking Interfaces.

volves a process of 'rationalization' or abstraction toward a 'template' which leaves the final customization of the behaviors for the actor at runtime. This both motivates our inclusion of execution components beyond the interface itself to translate the plan into action in the use environment (cf. Figure 2), and points to the importance of the task model as a means of communication between user and system. It also permits tasking at an abstract enough level to overcome many of the objections that Vicente [20] raises against static, prescriptive 'scripting' approaches to task-based instruction, while adhering to one of his central tenets: that designs need to be 'finished at run time' by the executing agent.

The second requirement of a task model for a tasking interface is that the tasks represented must be drawn from the way operators think about their domain. That is, the labeling conventions used by the interface and the system must be the same as those used by the human operators. 'Tasks' may be drawn from training materials, operator interviews, etc., but the resulting model should be intuitive and easily readable by any experienced operator.

Finally, the task model must also be understandable by all automation systems using it. It should serve the same function as a football team's playbook—each player knows what he is supposed to do when a given play is called, coordination is an emergent function stemming from all players sharing the same playbook and complicated behaviors can be activated rapidly by use of their 'labels'.

Our task modeling representation is based on a PERT-chart like approach developed initially by McDonnell-Douglas for use in their Pilot's Associate and Rotorcraft Pilot's Associate programs [7]. One possible enhancement to our approach would be shifting to a representation that explicitly models the distinction between goals and the plans that achieve them, such as Geddes Plan-Goal Graph [15]. The plan-goal distinction is a natural one for users to make in communicating intent [16].

In the creation of our tasking interface, we have extended the operator's ability to 'call plays' by enabling him/her to interact directly with the task model, activating and combining tasks at various levels of decomposition. This capability is provided via the *Playbook GUI* described in the next section below. We have also provided a planning system which is capable of understanding the operator's commands and either evaluating them for performability or, developing an executable plan which obeys, yet fleshes out, the operator's instructions. This *Mission Analysis Component* is described in the following subsection.

Playbook Graphical User Interface

The human operator must have a method of inspecting and interacting with the task model, both to understand the possible actions which could be taken to achieve known goals and, more importantly, to declare those tasks, goals, partial plans and constraints s/he wishes the system to pursue. This interface must provide the operator with a method of "calling plays" as described above. Just as a quarterback can activate a complex set of behaviors by referring to a simple play name and/or spend additional time (if available) refining those behaviors by combining play elements or tweaking play parameters, so the operator should be able to flexibly interact with his or her automation via the Playbook GUI.

Some requirements which the Playbook GUI must fulfill include: (1) the set of "plays" (e.g., maneuvers, procedures, etc.) represented must be those any well-trained operator should know (thus making it intuitive to learn and use), (2) the general play 'templates' in the playbook can be composed and instantiated to create any specific mission plan, (3) The operator may select plays at various hierarchical, leaving the lower levels to be selected and composed by the MAC, and (4) operators may either require or prohibit the use of specific plays or of specific resources within a play. This makes true mixed initiative planning possible and makes the tasking of a UCAV much like providing instructions to a human wingman.

One Playbook GUI is presented in Figure 3. While we began with the development of a direct viewing and manipulation tool for the task model itself, some limitations of this interface are clear. The underlying links to a task model permit a wide variety of interfaces. The specific attributes of the GUI must be driven by the uses to which it is put. As a simple example, a pre-mission planning tool will require much more flexibility and precision in visualizing and interacting with the emerging plan, while an in-flight tasking tool will be constrained to be more simple—perhaps an attenuated menu of only those tasking options currently relevant. Further, while a direct presentation of the task model (as illustrated in Figure 3) generally provides the most detailed and precise interaction with the emerging plan, it is not the most familiar or efficient presentation for many pilot planning tasks. For UCAV tasking, potential users have told us that interacting with the underlying task model indirectly via a map tool and a simple timeline view would be more intuitive, though the detailed task model tools should be retained for more complex mission specifications. We have included these concepts in the prototype tasking sys-

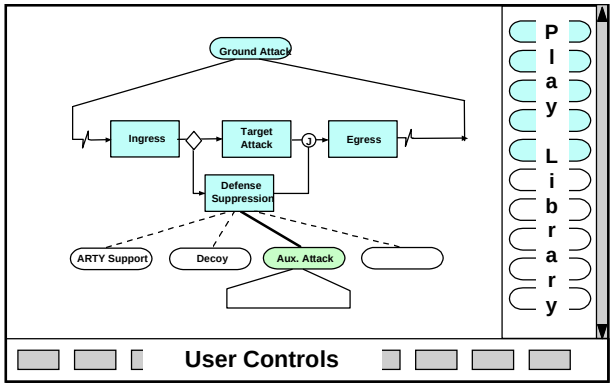


Figure 3. One possible instantiation of a playbook GUI.

tem described below.

Mission Analysis Component

The Mission Analysis Component (MAC) integrates projective planning technology with a human tasking mechanism. MAC operates over full or partial mission plans provided via the Playbook GUI, performing two sub-functions: (1) analyzing the operator's plan for feasibility and goal achievement by verifying constraints and (2) automatically generating candidate completions of partial mission plans in keeping with the requirements and prohibitions imposed by the pilot.

The hierarchical, typed representation of plays in the playbook simplifies choices for plan completion by limiting the types of plays that are useful at each level. The MAC uses a hierarchical task network planner [3] in conjunction with constraint propagation techniques [6,5] to perform the two functions described above. In broad outline, the structure of a UCAV mission is known before the planning has begun. Rather than rediscover this outline (a more traditional AI planning approach), the MAC must manage the resources, deadlines, etc. between alternative operational refinements during the development of the plan. The MAC represents these limited quantities as constraints on and between individual plan operators that are maintained by a constraint management engine. These plan operators are, however, sequenced and composed by a hierarchical task network planning algorithm in conjunction with a human operator.

The decomposition planner continuously determines the feasibility of the current partially specified plan. The shared task model, resident as a plan library usable by the MAC, specifies constraints on tasks and the subtasks they expand into. As a plan is built, constraints propagate both up, from sub-tasks to tasks, and down, from tasks to their expansions. This constraint propagation process enables the planner to identify flaws in the plan early. For example, if currently instantiated tasks require more fuel than will be available for the entire mission, the planner will detect the infeasibility and either automatically backtrack, if planning automatically, or notify the user, if evaluating a plan refinement s/he suggested.

Concurrently with checking for feasibility, the MAC fleshes out the abstract (non-primitive) tasks in the plan. When the MAC identifies a non-primitive, it decomposes by finding and then applying one or more subtask methods. In the case that more than one method applies, the planner develops the alternative paths. Infeasible alternatives are pruned from consideration. The GUI then displays the consequences of planning decisions to the user, who can retract previous choices, or make better-informed decisions from among the available, feasible choices.

In combination, feasibility checking and automatic plan expansion make it possible for the MAC to generate effective plans with a minimum of user involvement. Continual feasibility analysis minimizes the effort expended on dead-ends while encouraging the user to specify the mission critical aspects of the operation as early as possible. Once these are stipulated, the development of the plan can be left entirely to the MAC with the assurance that it will produce a plan that is both feasible within its constraint knowledge and in keeping the operator's stipulations. If time permits (or lack of trust demands), the user may provide increasingly detailed instructions by selecting among available plan alternatives, down to the lowest level primitive operators supported by the tasking interface.

Integrating Pre-existing Tools—Route Planning

A benefit of the architecture we have produced is that it enables incorporating pre-existing tools into the task-based environment that the interface provides. The effect is to provide task-sensitive, intelligent behavior in tools that were not built to include these capabilities. This feature is illustrated in our work with UCAV mission planning by incorporating a Honeywell route planner.

The route planner generates optimal (i.e., lowest cost) paths through digital models of actual terrain. The definition of optimality can be tuned to the application. Prior to incorporation into our tasking interface, a human operator had to deduce information about route start and endpoint parameters from mission objectives and partial plans (perhaps representing multiple alternatives) and then program them by hand into the route planner. Similarly, results from multiple route plans had to be interpreted and folded back into the planning process.

The structure of the MAC provides a straightforward way to incorporate specialty problem solvers like the route planner. We encapsulated the route planner within parameterized plan operators like a `follow_terrain` task. When that task is selected (by either the MAC or the user) bindings for start and endpoints of the route segment are inherited from higher level tasks and passed to the route planner along with cost parameters appropriate to the task (e.g., appropriate weightings on flight time, fuel requirements, exposure and threat levels, etc.). The route planner develops an optimal route within these parameters and returns it to the MAC along with new requirements on flight time and fuel usage. The MAC will determine if these new requirements violate constraints on the plan. If so, the route planner's actions will be undone. If no acceptable route can be found, the user will be notified.

Integration and Operation

To be useful in the real world, our tasking interface must bridge the traditional gap between analysis and execution. Analytic capabilities developed in previous AI planners have used simple, abstract task representations and have assumed correct execution. Reactive planning and execution systems are essentially high level programming languages. Reactive systems coordinate the behaviors of sensors and effectors to handle contingencies—but without support for analysis of correctness. Honeywell is currently at work on

an architecture, called CIRCA [12], that integrates projective and reactive planning with control actuation. CIRCA bridges the analysis/execution gap by synthesizing provably correct reaction programs to achieve high-level goals. To date, however, little thought has been given to how an operator will provide those goals. Our tasking interface fills that gap.

At the bottom layer of our architecture, advanced control algorithms provide for continuous vehicle control. These algorithms provide several levels of functionality. The highest level is the ability to fly specific flight segments (e.g. close or loose formation, "pop-up" for weapon release, rendezvous). At a lower level are guidance functions like waypoint steering and terrain following. Finally, at the lowest level are the attitude and rate stabilization control functions. The plan created jointly by the human operator and the MAC, along with reactive adaptations provided by the event handling component, gives these control functions the instructions they need. The result is a seamless ability to task and control the automation functions in a wide variety of ways at the human's discretion.

USAGE SCENARIO

The following scenario how a user might interact with the tasking interface prototype we have developed to plan a UCAV mission. Current and emerging UCAV interfaces either require operators to remotely control the aircraft via a dedicated cockpit mockup, or they rely on very high level behaviors (e.g., Close Air Patrol circuits or waypoint-designated routes) which can be easily but inflexibly commanded. The first approach provides high predictability and adaptiveness, but at the cost of very high operator workload; the second approach minimizes operator workload and provides highly predictable behavior, but only at the cost of adaptiveness. Neither approach is sufficient for usefully placing one or more UCAVs at the disposal of an operator who is concurrently piloting his/her own aircraft.

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

Our prototype tasking interface as illustrated in Figures 4-9. The interface illustrated in these figures *is* our current Playbook GUI. This interface runs in conjunction with a prototype MAC, communicating via a Shared Task Model. The MAC interacts with a specialty route planning algorithm. Plans produced by this tasking interface can be 'flown' in simulation using realistic control algorithms developed by Honeywell's Guidance and Control group.

Figure 4 (and, in larger format, Figure 9) shows the five primary regions of the Playbook GUI. The upper half of the screen is a *Task Composition Space* that the plan composed thus far. At startup, this area presents the three top-level tasks (or 'mission types') the system currently knows about: Interdiction, Airfield Denial, and Suppress Enemy Air Defenses (SEAD). The lower left corner of the interface is an *Available Resource Space*, currently presenting the set of aircraft available for use. The lower right corner contains an interactive *Terrain Map* of the area of interest. As noted above, some planning interactions can be more easily performed or understood via direct, graphical presentation against the terrain of interest. The space between these two lower windows (empty at startup) is a *Resource in Use Space*—once resources are selected for use, they will be moved to this workspace, where they can be interacted with in more detail. Finally, the lower set of control buttons is always present for interaction with the system. This includes options such as "Finish Plan" for handing the partial plan off to the MAC for completion and/or review, "Final Route" for accessing the route planner to create a route within constraints specified, "Print" and "Show Schedule" for obtaining a Gantt chart timeline of the activities planned for each actor, etc.

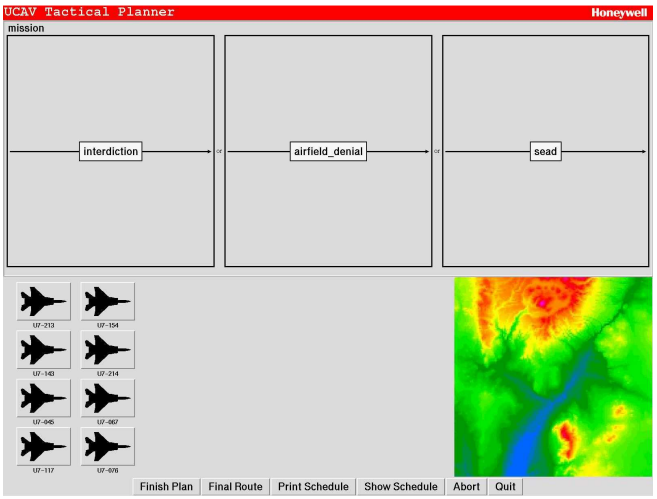
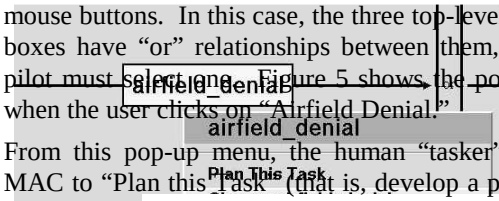


Figure 4. The UCAV Tasking Interface prototype at startup.

The mission leader would interact with the tasking interface to, first, declare that the "task" for the day was "Airfield Denial." Most interactions with the Playbook GUI are via mouse buttons. In this case, the three top-level mission task boxes have "or" relationships between them, meaning the pilot must select one. Figure 5 shows the pop-up window when the user clicks on "Airfield Denial."



From this pop-up menu, the human "tasker" can tell the MAC to "Plan this Task" (that is, develop a plan using this task) or indicate that s/he will "Choose airfield denial" as a task that s/he will flesh out further. The pop-up menu also contains a context-sensitive list of subtasks that the tasker

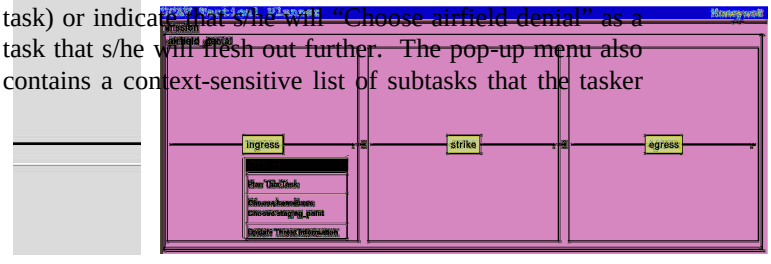


Figure 5. P

Figure 7. Second-level expansion of 'Airfield Denial' task.

can choose to include under this task (only one is appropriate at this level: “Suppress Air Defenses”).

At this point, having been told only that the task for the day is “Airfield Denial,” a team of trained human pilots would have a very good general picture of the mission they would fly. Similarly, our interface (via the MAC and the Shared Task Representation) knows what a typical airfield denial plan consists of. But just as a leader instructing a human flight team could not leave the instructions at that, so the tasker is required to provide more information to instantiate the high level task. Here, the tasker must provide the items listed in the next block in the pop-up window in Figure 5 (e.g., a target, a homebase, etc.) and, as for all plans, the specific aircraft to be used. S/he selects aircraft by clicking on the available aircraft resources and moving them to the Resources in Use Area. By clicking on “Choose Target”, the user activates the Terrain Map and a subsequent click there will designate a target location. The results of these actions are illustrated in Figure 6.

The human leader could provide substantially more detail (such as route, munitions, roles for the wingman, etc.) but s/he could also hand the task off to intelligent human team members at this point and let them develop the plan. The tasker can do this as well, handing the task to the MAC via the “Finish Plan” button. We will assume, though, that the user wishes to provide more plan specifications.

The final option on this pop-up menu allows the user to “Update Threat Information” for use in planning from a dynamically maintained, external database.

Both the tasker and the interface know, thanks to their Shared Task Model, that any Airfield Denial plan must consist of Ingress, Strike and Egress subtasks, in that order. “Airfield Denial” may also include a Defense Suppression subtask that runs in parallel with Strike. After the tasker selects “Choose Airfield Denial”, the Task Composition Space is reconfigured as in Figure 7 to show the next level of options. The small arrows between the task blocks indicate that all subtasks are sequential.

To provide detailed instructions about how to perform the Ingress task, the tasker must choose it, producing the “generic” Ingress task shown in Figure 8. Note that this is not a default method of doing “Ingress” so much as a generic, uninstantiated template—corresponding to what a human expert knows about how Ingress can or should be. A trained pilot knows that Ingress can be done either in formation or in dispersed mode and, in either case, must involve a “Take Off” subtask followed by one or more “Fly to Location” subtasks. Figure 8 illustrates the GUI after the user selects “Formation Ingress”. The MAC automatically fills in the set of steps it knows must be accomplished to do Ingress in formation— that all aircraft must takeoff and then fly to a specified location. In the event that there are further specifications the user can or must supply, the MAC

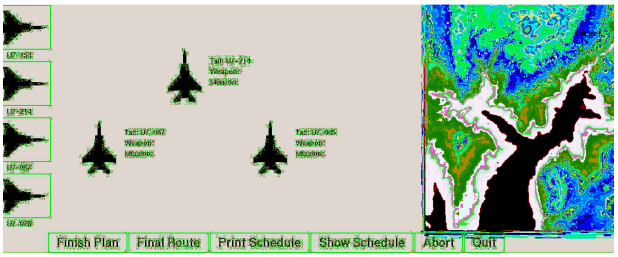


Figure 6. Illustration of resource and target selection.

will generate them in context-sensitive pop-up windows (e.g., specifying the airfield to take off from.)

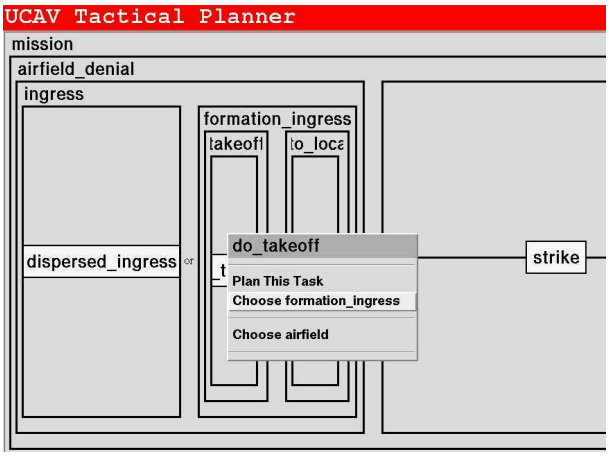


Figure 8. Stipulating subtasks under “Ingress”.

The user can continue to specify and instantiate tasks down to the “primitive” level where the subtasks are behaviors the control algorithms (see Figure 2) in our simulator can be relied upon to execute in flight. Alternatively, at any point after the initial selection of mission task and its required parameters, the tasker can hand the partly developed plan over to the MAC for completion and/or review by means of the “Finish Plan” button. In extreme cases, a viable “Airfield Denial” plan could be created with as few as five choices (select aircraft, target, homebase, staging and rendezvous points). If the MAC is incapable of developing a viable plan within the constraints imposed, (e.g., if the user has stipulated distant targets that exceed aircraft fuel supplies) MAC will inform the user of these difficulties via pop-up windows.

Figure 9 shows the completed plan provided by the MAC after the stipulation of the Ingress task as described above. The user has asked for a Final Route generated by the route planner on the basis of the MAC’s inputs. This output is presented in the Terrain Map window.

ACCOMPLISHMENTS AND CONCLUSIONS

Our prototype interface builds plans for 3 types of multi-UCAV missions (corresponding to 3 top-level tasks). The prototype was implemented with SICStus Prolog, Tcl/Tk, and the daVinci graph visualization tool [4]. We implemented our own HTN planner in Prolog and used the SICStus Constraint Logic Programming for Finite Domains module to maintain the constraint store. We built the illustrated version of the Playbook GUI using SICStus's embedded Tcl/Tk interpreter and daVinci.

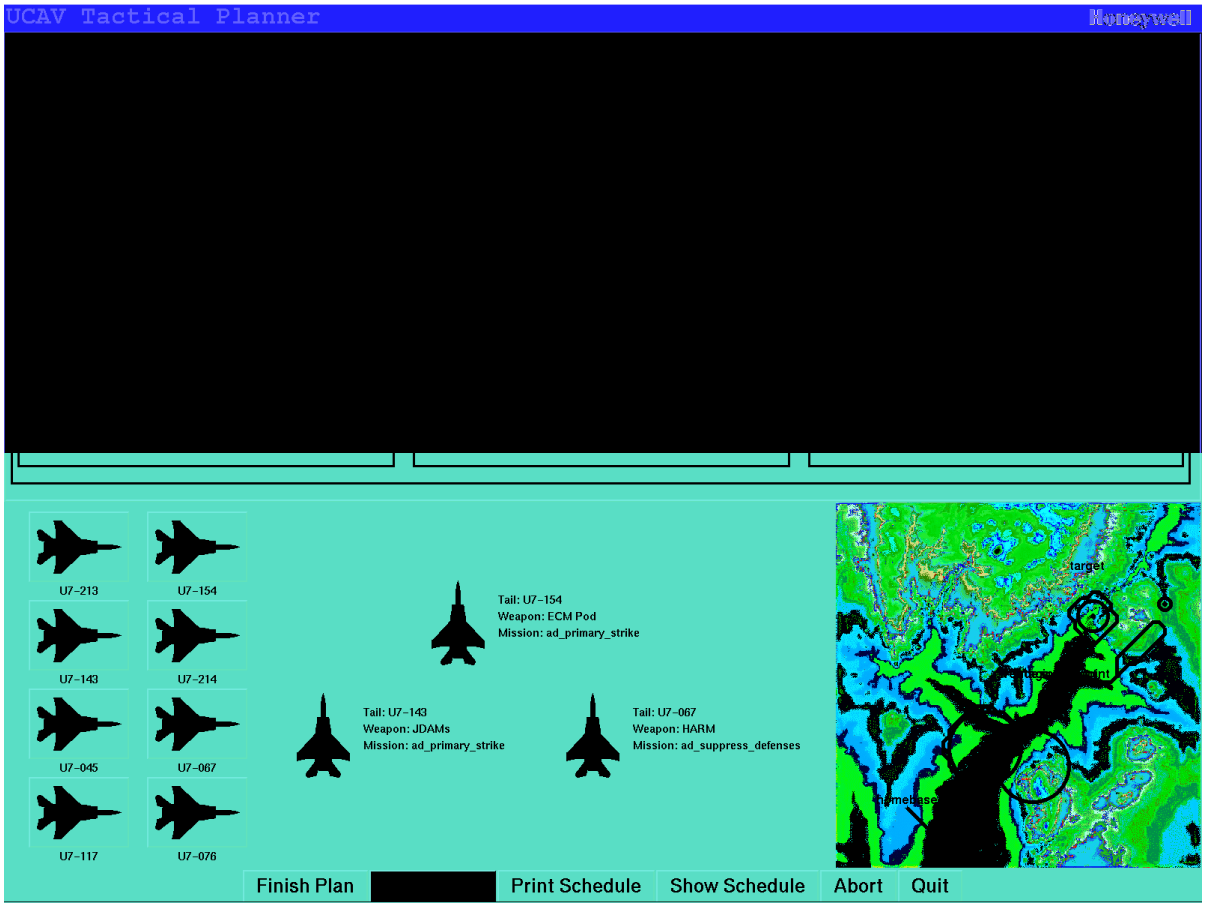


Figure 9. Finished plan returned by the MAC after User specification of the Ingress task.

Other types of interfaces are entirely possible (see [11] for a more ambitious, though unimplemented example illustrating more sophisticated interactions with the MAC and shared task model) and we hope to explore their development as we carry the tasking interface concept into alternate domains and gain more experience with user interactions with this type of automation.

Our approach to tasking interfaces builds a bridge of flexibility between 'pure' direct manipulation interfaces where the user is always in charge but must incur the associated workload, and 'pure' intelligent agent automation where the user is free from undue workload but at the cost of surrendering control and predictability to a system that may not do what s/he wants. While operators of complex systems are rarely comfortable giving over authority to automation at all times, they are quite willing to use it when helpful. Tasking interfaces are a method of allowing the operator to remain fully in charge, yet of enabling almost full autonomy for an aiding agent. Perhaps by requiring (and enabling) automation to behave more like an intelligent subordinate, operators will be more tolerant of its weaknesses and more willing to let it show its capabilities in some settings. Through use, then, users may become more familiar with their automation's strengths and weaknesses and, ultimately, better at using it when it is needed.

ACKNOWLEDGEMENTS

This work was funded by a Honeywell Initiatives Grant. The authors would like to that Dan Bugajski, Don Shaner and John Allen for their help in the development and implementation of the ideas presented.

REFERENCES

1. Billings, C. *Aviation Automation: The search for a human-centered approach*. Lawrence Erlbaum: Mahwah, NJ, 1997.
2. Boy, G. *Cognitive Function Analysis*. ABLEX; Stamford, CT, in press.
3. Erol, K., Hendler, J. and Nau, D. UMCP: A sound and complete procedure for hierarchical task network planning. In Hammond, K. (Ed.) *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*, (Los Altos, CA, 1994), 249-254.
4. Frohlich, M. and Werner, M. The daVinci graph visualization tool, 1997. Available at http://www.informatick. unibremen.de/~davinci.
5. Hentenryck, P. *Constraint Satisfaction in Logic Programming*. Cambridge, MA: MIT Press, 1989.
6. Jaffar, J. and Michaylov, S. Methodology and implementation of a CLP system. *Proceedings of the Fourth International Conference on Logic Programming*. (Cambridge, MA: MIT Press), 1987.
7. Keller, K. and Stanley, K. Domain specific software design for decision aiding. *Proceedings of the AAAI Workshop on Automating Software Design.* (NASA Ames Research Center, July, 1992), 86-92.

8. Layton, C., Smith, P, and McCoy, E. Design of a cooperative problem solving system for enroute flight planning: An empirical evaluation. *Human Factors, 36*(1), 1994, 94-119.

9. Maes, P. Agents that Reduce Work and Information Overload. *Communications of the ACM, 37(7),* 1994. 31-40.

10. Miller, C. & Funk, H. Task-based Interface Management: A Rotorcraft Pilot's Associate Example. *Proceedings of the AHS Crew Sytems Technical Specialists Meeting*, (Philadelphia PA, September 1997).

11. Miller, C. and Goldman, R. 'Tasking' Interfaces: Associates that know who's the boss. *Proceedings of the 4th USAF/RAF/GAF Conference on Human/Electronic Crewmembers, (*Kreuth, Germany, September 1997).

12. Musliner, D., Durfee, E., & Shin, K. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics, 23*, (1993), 1561-1574.

13. Riley, V. Operator reliance on automation: Theory and data. In R. Parasuraman and M. Mouloua (Eds.), *Automation and Human Performance: Current Theory and Applications.* Lawrence Erlbaum: Hillsdale, NJ, 1996, 19-36.

14. Scacchi, W. Modeling, Integrating and enacting complex organizational processes: Approach and experience. In A. Seth (Ed.), *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions*, (Athens, GA, 8-10 May 1996). 18-23.

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

16. Shattuck, L. *Communication of Intent in Distributed Supervisory Control Systems*. Unpublished dissertation. The Ohio State University, Columbus, OH. 1995.

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

18. Shneiderman, B., Direct manipulation for comprehensible, predicatable, and controllable user interfaces, *Proceedings of the ACM International Workshop on Intelligent User Interfaces '97,*(New York, NY, 1997) 33-39.

19. Vicente, K. J. Improving dynamic decision making in complex systems through ecological interface design: A research overview. *System Dynamics Review, 12*, 1996, 251-279.

20. Vicente, K. J. *Cognitive work analysis: Towards safe, productive, and healthy computer-based work.* Erlbaum: Mahwah, NJ, in press.