

# “Tasking” Interfaces; Associates that Know Who’s the Boss

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

## 1. SUMMARY

Despite substantial technological success, “associate” systems continue to suffer from a fundamental, sociological problem—namely, that human operators of advanced automation are used to being in charge. We have recently begun adapting techniques from our associate research to the construction of “tasking” interfaces which enable the kind of interaction operators are used to having with intelligent, informed subordinates. Instead of being autonomous, or even truly mixed-initiative, “tasked” systems are always subordinate—but they know enough about the tasks in the domain that instructing them is vastly easier than instructing traditional automation systems. We use a shared task model to give advanced automation systems (e.g., Unmanned Air Vehicles—UAVs) the same task and goal understanding that the human has. When combined with a planner, the resulting system permits ‘tasking’ at all the various levels an intelligent subordinate should be able to accept: exhaustively specified plans to be performed exactly, partial plans which leave the planner free to create any full plan which includes those pieces, constraints which require the planner to stay away from certain methods or resources, or high-level goals for the automation to achieve however it thinks best. The net result is a human-machine system which is almost as capable as an associate, and which reduces human workload almost as much, yet which leaves the human more in control than an associate does.

## 2. INTRODUCTION—THE PROBLEM

In the approximately 12 years since the first associate systems were conceived and development work was begun, workers in the field of human-electronic crewmember systems have achieved an impressive array of technological successes. Numerous working prototypes have been demonstrated and are beginning to find their way into common use. The field has expanded far beyond its initial beginning with associates for fighter pilots, to encompass commercial aviation, automobile driving, oil refinery control room applications, etc. The U.S. Army is on the verge of flight testing its Rotorcraft Pilot’s Associate and even Microsoft’s latest Office™ products are shipping with a task-based, intent-tracked help system.

In spite of these technological achievements, experience has consistently shown that the concept of a human-electronic

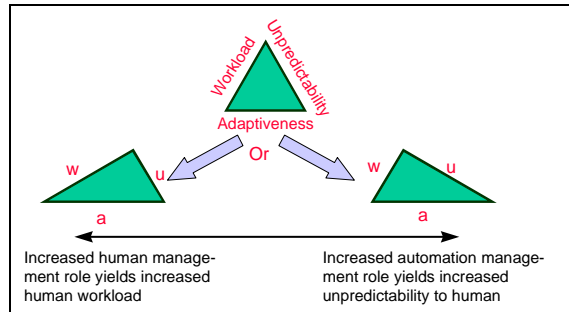


Figure 1. Conceptual view of the relationship between system adaptiveness, human workload and unpredictability to the human operator.

crewmember, as it has been implemented to date, suffers from a basic sociological problem. Namely, pilots (and other human operators of complex systems) want to remain in charge of the equipment they use. In developing the Rotorcraft Pilot’s Associate Cockpit Information Manager [1], we interviewed a variety of rotorcraft pilots and RPA designers to develop a consensus list of prioritized goals for a “good” cockpit configuration manager. Two of the top 3 items on the list were “Pilot remains in charge of task allocation” and “Pilot remains in charge of information presented.”

By definition [2], and for good reasons, an associate system shares responsibility, authority and autonomy over many cockpit behaviors with the human operator(s). This is especially true of the traditional associate behaviors of information management and adaptive automation. It is important to remember, however, that the motivation for creating associate systems which had the capability to share these tasks with the operator has always been to reduce operator workload and information overload [3]. While operators wish to remain in charge, and it is desirable that they do so, the simple fact is that in today’s complex systems, operators cannot be fully in charge of all system operations—certainly not in the same way they have been in earlier cockpits and workstations.

Conceptually, the problem can be presented as in Figure 1. 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. The implication of this view is that for any increase in adaptiveness (that is, the ability of the human-machine system to perform in an appropriate, context-dependent manner in different situations) there must be an accompanying increase in either human workload (the amount of physical, attentional or cognitive “energy” the human must exert to use the system) or in unpredictability for the human operator (inability of the human to know what the automation will do at any given time). Since adaptiveness is generally the goal of added complexity (though systems can be complex without achieving this goal), this is equivalent to saying that any increase in human-

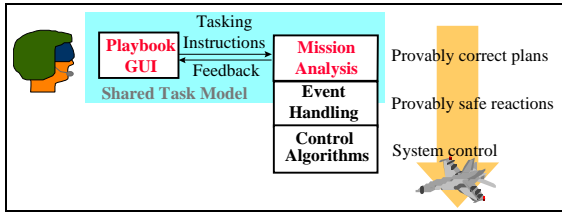


Figure 2. General Architecture for Tasking Interfaces.

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 human workload, or (2) the added complexity must be managed by automation, resulting in increases in unpredictability to the human. These alternatives represent endpoints on a spectrum; many intermediate points are possible.

Associate systems have chosen to address the problems incurred by increased adaptiveness by adding an extremely sophisticated suite of cockpit automation. Thus, they are near the “automation” end of the spectrum of solutions. When successful, they do obtain the benefit of greatly enhanced system adaptiveness with little or no increase in human workload. But they continue to encounter the socio-technological hurdle of diminished control and increased unpredictability for the human operator.

In the remainder of this paper, we present initial work on a solution which allows human operators to interact with advanced automation at a variety of levels. While this does not eliminate the dilemma presented in Figure 1, it mitigates it by allowing operators to flexibly choose various points on the spectrum for interaction with their automation. 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’ an intelligent, knowledgeable subordinate. The notion of tasking gives the operator the same type of control s/he currently has over tasks delegated to subordinates, thus it provides him or her with a greater degree of control than a full mixed-initiative associate system does, while allowing low workload interactions in situations where they are desirable.

### 3. PROPOSED SOLUTION— TASKING INTERFACES

Figure 2 presents our general architecture for tasking interfaces. The primary components are a Graphical User Interface (GUI) and a Mission Analysis component which 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 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 [4]. The Mission Analysis component is a projective planning system which is capable of understanding these instructions and (a) evaluating them for feasibility and b) expanding them (where necessary) to produce fully executable plans. Once an acceptable plan is created, it is passed to an Event Handling component which is itself a reactive planning system capable of making moment by moment adjustments to the plan to enable execution. The event handling component then passes these instructions to control algorithms which actually effect behaviors in the controlled system automation. Since the Playbook GUI, Mission Analysis

component and the Shared Task Model are components unique to the construction of a tasking interface, as well as the focus of our current research, they will be described in more detail in separate subsections below.

#### 3.1 Shared Task Model

The enabling technology for a tasking interface is the ability to code, track and dynamically modify the plans, goals, tasks and objectives which one or more humans may have in operating their automation and equipment. By explicitly representing such a “task model” in a format that is both familiar and recognizable by a human operator *and* which is interpretable by a knowledge-based planning system, we gain a level of coordination between human and system beyond what was previously possible. Work on the U.S. Air Force’s Pilot’s Associate pioneered such task models in two different formats: a plan-goal modeling technique [5] and a Task Network representation based, ultimately, on PERT chart notations used in business planning [6].

These modeling techniques share several critical attributes, as illustrated in Figure 3. First, they are organized via hierarchical decomposition—meaning that beneath each task or goal the next lower layer represents alternate methods of achieving that goal, down to some bottom layer of executable, primitive actions. Models have both breadth and depth. Their breadth is the range of task domain which they cover. A task model for interactions with a weapon system is “narrower” than a model which covers all flight operations. Their depth is a measure of the degree of decomposition or “granularity” on the lowest level of actions represented—thus a model of flight operations which only represents major flight phases (e.g., take off, ingress, attack, etc.) is “coarser” than one which decomposes these actions still further.

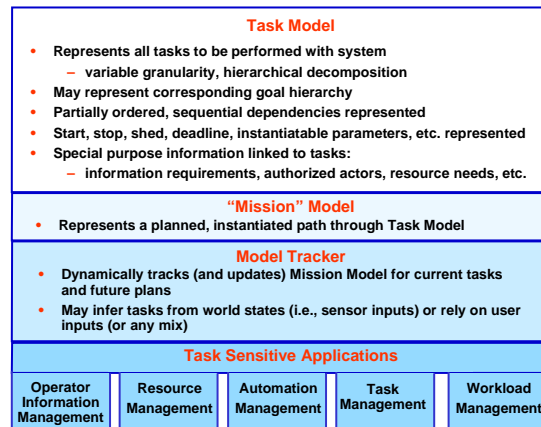


Figure 3. The various uses and instantiations of task models in associate systems.

Task models have both static and dynamic variations. In their static variation, they list all possible tasks that can be performed in this human-machine system (to some level of depth). From among this static set of possible tasks, certain tasks can be flagged and combined a priori as the expected states for some portion of the future (e.g., the mission plan). Static mission models may then be “tracked” or updated dynamically during operation to represent the current, unfolding set of tasks the human-machine system is actually engaged in—both indicating when various tasks in the mission plan become active and/or when the human operator has chosen to do perform some task which was not explicitly planned for this mission (e.g., react to threats).

The tasks represented in the model are (or should be) drawn from the way operators think about tasks, goals, etc. in their domain. They may be drawn from training materials or operator interviews. In any event, the resulting model should be easily readable by a trained operator in the domain. But tasks in the model must be interpretable by all automation systems which will use them (something like a football team's playbook—each player knows what he is supposed to do when a given play is activated and coordination is an emergent function stemming from their all sharing the same playbook). Thus, by 'calling a play' (that is, declaring that a certain task is to be accomplished), the operator can task automation subsystems to behave appropriately for the performance of that task.

In the creation of our tasking interface, we have extended the operator's ability to 'call plays' by providing them the ability to interact directly with the task model, activating 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, when they are provided at a level higher than is executable by automation, of developing an executable plan which obeys, yet fleshes out, the operator's instructions. This *Mission Analysis Component* is described in the following subsection.

### 3.2 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. Through this interface the operator (e.g., pilot) will graphically construct a full or partial plan for the mission specifying the tasks (or "plays") to be performed and goals to be accomplished. Some requirements for the Playbook GUI we are constructing include: (1) the set of "plays" (e.g., maneuvers, procedures, etc.) represented will be those that any well-trained operator should know (thus making it intuitive and easy to learn and use), (2) the general play 'templates' in the playbook can be composed and instantiated to create any specific mission plan, and (3) The operator may select plays to be used at various levels in the hierarchical decomposition of the mission plan, leaving the remainder to be selected and composed by the Mission Analysis Component (see below), either requiring or prohibiting the use of specific plays. This makes true mixed initiative planning a

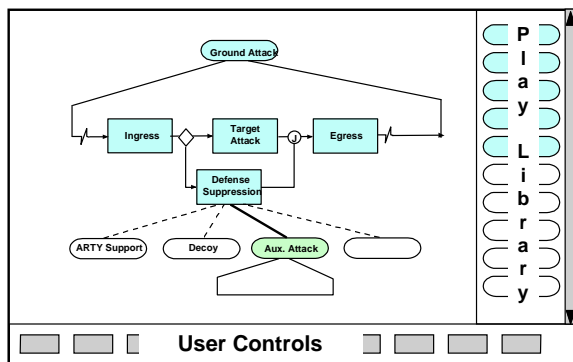


Figure 4. One possible instantiation of a playbook GUI.

possibility and will make the tasking of a UAV much like providing instructions to a human wingman.

A hypothetical example of a Playbook GUI is presented in Figure 4. While we are only beginning the development of this tool, some factors are already clear. First, the underlying links to a task model both permit and require a wide variety of interfaces. The specific attributes of the GUI will be driven by the uses to which it will be 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 use something as simplified as a highly-attenuated menu of only those tasking options which are relevant at the current moment. Another factor which is becoming obvious in our initial development activities is that, while a direct presentation of the task model (as is illustrated in Figure 4) generally provides the most detailed and precise interaction with the emerging plan, it is neither the most familiar nor the most efficient presentation for many pilot planning tasks. Interaction with the underlying model via a map-based presentation and a simple timeline view are both being considered to address this problem.

### 3.3 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 means of verifying constraints and (2) automatically completing partial mission plans (or suggesting candidate completions) 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 which are useful at each level. Plays contain information about their expected and desired effects. This information is used by projection algorithms in the MAC to analyze the plan for correctness (i.e., will it achieve its stated goals if executed successfully?). Constraint propagation techniques are used to coordinate the individual plays chosen by the pilot or the system into coherent mission plans (e.g., once a target is stipulated for one phase of the plan, it is propagated to the other phases automatically). The MAC module we are developing draws on work in AI planning, but must be part of an approach bridging the traditional gap between analysis and execution. Analytic capabilities developed in previous AI planners have used simpler, more abstract task representations and have made substantial assumptions about correct execution. Reactive planning and execution systems (such as those used in robots) are essentially high level programming languages used to coordinate the behaviors and states of sensors, effectors and to handle contingencies—but without support for analysis of correctness. Honeywell is concurrently at work on an integrative architecture for projective planning, reactive planning, and control actuation, called CIRCA [7] which bridges this gap by synthesizing provably correct reaction programs in accordance with known goals. To date, however, little thought has been given to how an operator will provide those high-level goals. Our work fills that gap.

### 3.4 Integration and Operation

At the bottom layer of our architecture, advanced control algorithms provide for actual moment-to-moment vehicle control. The control 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 resulting "plan" created jointly by the human operator using the Playbook GUI and by the MAC's reviewing or fleshing out the human's instructions, along with reactive adaptations provided by the event handling component, gives these control functions the instructions they need to know which behaviors to execute. The result is a seamless ability for the operator to task and control the automation functions in a wide variety of ways depending on the human's available time and degree of trust.

#### 4. A TASKING INTERFACE FOR UAVs

We have recently begun work on a proof-of-concept demonstration of this tasking interface architecture. We have chosen to work in the domain of Unmanned Air Vehicles (UAVs) both because this domain is of interest to Honeywell and its customers and because the tasking of UAVs by human operators already engaged in highly demanding activities (e.g., aircraft or helicopter pilots) is a good example of the situation depicted in Figure 1 above. Current and emerging UAV interfaces either require operators to remotely control

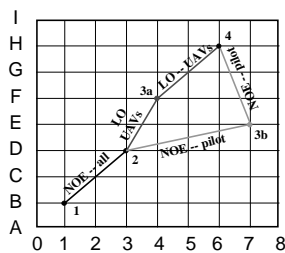


Figure 5. Hypothetical route plan.

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 commanded but not modified. The first approach provides a high degree of predictability and adaptiveness, but only 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 of these approaches is sufficient for placing one or more UAVs at the disposal of an operator who is concurrently engaged in the piloting of his own aircraft. To make coordinated operations between a human pilot and UAV(s) feasible demands increases in adaptiveness without substantial increases in either unpredictability or pilot workload.

Building on prior control algorithms and simulation work supporting a scenario of one piloted F-16 with two unmanned F-16 "wingmen", we have begun developing a tasking interface to enable a human pilot to lay out a mission plan for the UAV's. This interface will support the stipulation of goals, partial plans, full plans and constraints for each of the UAVs

either separately or in conjunction. We have begun our work by concentrating on a ground-based tasking interface due to its lighter demands on pilot, simulation and interface design. However, we believe that, with suitable modifications to the GUI, this approach will be suited to in-flight tasking as well.

To date, we have used a variation on the PERT chart-based task network representation as pioneered by McDonnell-Douglas. A portion of this representation for the high-level task of "ground attack" is presented in Figure 4. While definition of the Playbook GUI is proceeding at this time, we will step through a representative example of interaction between the human operator, the Playbook GUI and the Mission Analysis component below.

The human "leader" of the pilot + UAVs team (presumably the pilot himself) would interact with the tasking interface to, first, declare that the "mission task" for the day was "Ground Attack." Having declared only that much to a team of trained human pilots, the team would have a very good general picture of the mission they would be flying; similarly, our interface (via the MAC) knows what a typical ground attack plan consists of. But just as a human leader instructing a flight team could not leave the instructions at that, so the human "tasker" is required to provide a bit more information to instantiate and bind the high level task. In this case, the tasker *must* provide at least the specific target of the ground attack. S/he could provide substantially more detail (such as take off time, route, munitions, roles for the wingman, etc.) but s/he could also hand the task off to intelligent team members at this point and let them work out the best plan they can come up with. The tasker can do this as well, using our interface to hand the task to the MAC with only this information. For our example, we will assume that the tasker wishes to provide more plan specifications.

Both the tasker and the interface know that any Ground Attack plan must consist of Ingress, Target Attack and Egress subtasks, in that order. It may also include a Defense Suppression task which would run in parallel with Target Attack. These relationships are depicted in the task network in Figure 4: conditional branches are indicated by diamonds, sequential relationships are depicted left to right and parallel tasks are depicted next to each other vertically with a round "join" node showing that both parallel tasks must be completed before the next task can begin. Figure 4 also shows the tasker selecting among alternative methods of performing the "Defense Suppression" task.

Assume that the tasker wishes to provide detailed instructions about how the Ingress task is to be performed. His instructions are presented graphically in Figure 5. The tasker wishes the whole team to fly Nap of the Earth (NOE) from waypoint 1 to 2 and then to split, with the two UAVs flying a Low Observable (LO) route to waypoint 3a and on to 4 (perhaps serving as a decoy), while the pilot will fly NOE to waypoints 3b and on to rejoin at 4.

In order to task the UAVs to perform in this fashion, the leader must begin by expanding the Ingress task in the plan developed in Figure 4 above. Upon doing so, he would receive the “generic” Ingress task representation shown in Figure 6. It is important to note that this is not a default method of doing “Ingress” so much as it is a generic, uninstantiated method—corresponding loosely to what a human operator knows about how Ingress can or should be done in general. That is, the trained pilot knows that in order to accomplish Ingress, they will need to all Take Off, will probably need to Assemble, will certainly need to Fly to Objective and then Prepare for Split (assuming they assembled in the first place). For our example, we’ll assume that the leader doesn’t wish to place constraints on how the Take Off and Assemble tasks are to be performed (that is, he will leave the planning of these tasks up to the MAC), but that he does wish to mandate that the “Fly to Objective” task be performed in a specific way. We are using a dotted line convention in these figures to indicate no constraints imposed, by either human or MAC, on the conduct of this task and grey lines plus shading to indicate that the pilot has imposed constraints. Tasks are represented by boxes, and the actor(s) associated with the task are indicated in the lower left corner of the box. Figure 6 also illustrates a pop up window via which the tasker can input required information for the task.

If the pilot expands the Fly to Objective task to begin to input his route constraints, he first gets a “generic” method of performing the task—as illustrated in Figure 7. This diagram says that one can fly to an objective by doing one or more (note the optional loop) Fly to Waypoint or Fly Air Corridor tasks. In Figure 7, the pilot has chosen a Fly to Waypoint task and is entering required information to indicate that all actors will fly to waypoint 2 at location D3.

Since the MAC realizes that the performance of this Fly to Waypoint task does not yet accomplish the parent task’s goal of reaching the objective (that is, of being at H6), MAC instructs the GUI to present another set of the generic options it knows are available to accomplish the goal. The developing task network is redrawn as in Figure 8. Here the first “Fly to Waypoint” task is included as a stipulated part of the plan, but the same set of choices is presented for the remaining flight segments, since the system knows that this is how “flying to an objective” can be accomplished. Again, the tasker could choose to leave the remaining specification to the MAC—in which case, MAC would develop a plan which incorporated the first flight or report that this was impossible. Instead, however, the tasker goes on to stipulate how the next segment is to be flown, indicating that the two UAVs are to fly to waypoint 3a by themselves.

If the tasker continues to stipulate the set of waypoints indicated in the hypothetical route plan in Figure 5 above, the resulting task network would look like Figure 9, with separate branches for the pilot and the two UAVs for waypoints 3 and 4. Figure 9 also illustrates the further expansion of the Fly to Waypoint tasks to illustrate how the tasker can impose con-

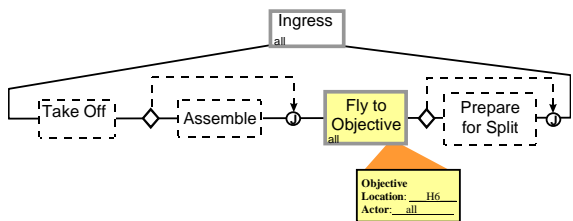


Figure 6. Expansion of Ingress Task.

straints on low level details about how the flying is to be done. By decomposing the “Fly to Waypoint” task down still further, the tasker can access attributes of the flying task: flight formation, system configuration, flight method, etc. Under each of these, there is a series of “Achieve” and “Fly”

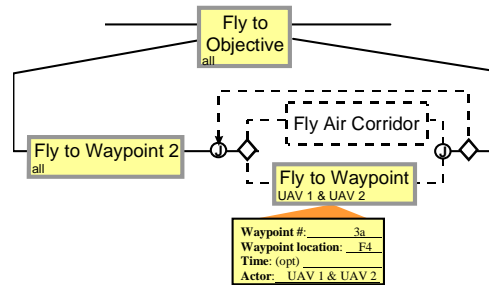


Figure 7. Expansion of Fly to Objective for second waypoint, UAVs only.

tasks which correspond to various options about how flight may be performed. We have expanded the “Achieve Flight Method” branch to illustrate the stipulation of NOE flight for all actors for the first flight segment. The remaining segments could be stipulated similarly. Note, though, that the tasker has made no constraints on the remaining flight parameters, leaving MAC free to plan these as needed to accomplish higher level mission tasks or obey other imposed constraints. Note too, that the flight “behaviors” depicted at the lowest level in Figure 9 are now at the highest level that the control algorithms for UAVs are currently capable of executing. Thus, we have reached the level at which model development can reasonably end.

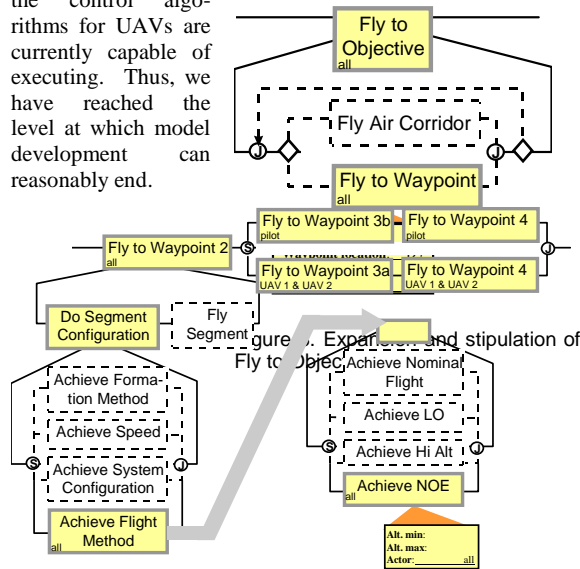


Figure 9. Expansion of Fly to Waypoint to flight parameters.

## 5. DIRECTIONS FOR FUTURE GROWTH

We are still in the design stages of our proof of concept tasking interface, but we are already identifying methods for improvement. First among these is the need to evaluate usability. We intend to allow pilots and engineers with military mission planning experience to review our prototype, but a thorough evaluation, ideally in comparison with traditional tasking techniques, would be more appropriate.

Although the focus of our initial research has been the development of the task model infrastructure and methods for human and computer sharing of it, experience indicates that we

need several interface improvements. First, we must integrate the task model view (presented above) of the plan with alternate views including a map-based tool (to provide better orientation and a more familiar method of creating mission plans), and a timeline view to provide better visualization of the temporal layout of mission. Second, we need to enable rapid, simple interaction with the model at multiple levels—taskers will frequently wish to specify a general plan with one very detailed constraint (e.g., ‘runway attack, but don’t use rockeyes,’ but with the current interface they can only reach that constraint by stepping through multiple layers of a detailed plan. Third, operators will wish to indicate that many tasks are to be done the same way (e.g., fly all waypoints after crossing the FEBA in NOE) but at present they must input this constraint separately for each task. We are now exploring the use of an object-hierarchy of tasks as a means of applying a constraint to multiple tasks within a class. Finally, we need a simple method of dictating negatives to the MAC—that is, stating that any method of accomplishing the task is acceptable *except* this one.

While we plan to address the above issues, further necessary developments are beyond our current scope: (1) Our current task network representation is weak in its coding of goals, and we believe that goals are a critical component of any tasking interaction [4]. Future work should explore representations which mix the sequential strengths of the task network with the goal-to-plan relationships of a plan-goal graph. (2) Tasking interfaces should not rely on a pre-defined set of task models. The operator should be able to create novel tasks and to store components of models which are useful. This indicates the need for a compositional tool and a library of stored models. (3) We believe that a “run-time” (e.g., in flight) version of the tasking interface is feasible and desirable. Such an interface would necessarily support a more narrow range of operator controls and modifications of the model. It should provide more rapid access to a tightly constrained set of alternate plans or modifications of high-level parameters within existing plans. To facilitate transfer of training, however, it should maintain at least some of the look and feel of the ground-based tool.

## 6. CONCLUSIONS

Our approach to tasking interfaces is intended to build a bridge from current systems to true “associates”. While pilots are rarely comfortable giving over authority to automation at all times and situations, they are willing to have it around for those times when it may be useful. Tasking interfaces are a method of allowing the pilot to remain fully in control, yet of enabling almost the full autonomy of an associate to plan and execute a high-level task whenever the pilot deems that that level of assistance is appropriate. The net result is a human-machine system which is almost as capable as an associate, and which reduces human workload almost as much, yet which leaves the human more in control than an associate does. Perhaps by requiring (and enabling) an associate to behave more like an intelligent subordinate, pilots will be more tolerant of their weaknesses and more willing to let them show their capabilities in tightly controlled settings. Through use, then, pilots may become more familiar with their strengths and, ultimately, more willing to tolerate them on a roughly equal footing in the cockpit.

## ACKNOWLEDGEMENTS

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

## REFERENCES

1. Miller, C. & Funk, H. (1997). “Task-based Interface Management: A Rotorcraft Pilot’s Associate Example.” In Proceedings of the AHS Crew Systems Technical Specialists Meeting. Philadelphia, PA; September 23-25.
2. Yadrick, R., Judge, C., & Riley, V. (1990). “Decision support and adaptive aiding for a battlefield interdiction mission.” in the *Proceedings of The Human-Electronic Crew: Is the Team Maturing*, Ingolstadt, Germany.
3. Banks, S. & Lizza, C., (1991). “Pilot’s Associate; A cooperative knowledge-based system application.” *IEEE Expert*, June 1991, 18-29.
4. Shattuck, L. (1995). *Communication of Intent in Distributed Supervisory Control Systems*. Unpublished dissertation. The Ohio State University; Columbus, OH.
5. Gedded, N. (1989). *Understanding Human Operators’ Intentions in Complex Systems*. Unpublished dissertation. Georgia Institute of Technology; Atlanta, GA.
6. McBryan, B. & Hall, J. (1994). “Engineering approach for Rotorcraft Pilot’s Associate Cognitive Decision Aiding System development.” In *Proceedings of the 13<sup>th</sup> AIAA/IEEE DASC*. Phoenix, AZ; Oct. 30-Nov. 1.
7. Musliner, D., Durfee, E., & Shin, K. (1993). “CIRCA: A cooperative intelligent real-time control architecture.” In *IEEE Transactions on Systems, Man and Cybernetics*, 23, 1561-1574.