High Assurance Human-Centric Decision Systems

Constance Heitmeyer*, Marc Pickett†, Len Breslow†, David Aha†, J. Greg Trafton† and Elizabeth Leonard*
*Center for High Assurance Computer Systems
†Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Washington, DC 20375
Email: {constance.heitmeyer, marc.pickett.ctr, len.breslow, david.aha, greg.trafton, elizabeth.leonard}@nrl.navy.mil

Abstract—**Many future decision support systems will be human-centric, i.e., require substantial human oversight and control. Because these systems often provide critical services, high assurance will be needed that they satisfy their requirements. How to develop "high assurance human-centric decision systems" is unknown: while significant research has been conducted in areas such as agents, cognitive science, and formal methods, how to apply and integrate the design principles and disparate models in each area is unclear. This paper proposes a novel process for developing human-centric decision systems where AI (artificial intelligence) methods—namely, cognitive models to predict human behavior and agents to assist the human—are used to achieve adequate system performance, and software engineering methods, namely, formal modeling and analysis, to obtain high assurance. To support this process, the paper introduces a software engineering technique—formal model synthesis from scenarios—and two AI techniques—a model for predicting human overload and user model synthesis from participant studies data. To illustrate the process and techniques, the paper describes a decision system controlling unmanned air vehicles.**

I. Introduction

Many future decision systems will be *human-centric*—i.e., require substantial human oversight and control. Because these systems often provide critical services, high assurance will be needed that they satisfy their requirements. Systems controlling autonomous vehicles constitute one important and growing class of human-centric decision systems which require high assurance. Currently, the largest deployed class of systems which control autonomous vehicles manage UAVs (unmanned air vehicles): the U.S. military alone is estimated to deploy over 7,000 UAVs, compared to less than 50 a decade ago [5]. Currently, these systems, which perform a range of challenging tasks including surveillance and targeting, are not entirely autonomous but remotely controlled by humans. In future years, systems managing autonomous vehicles are expected to be widely deployed in both military and non-military applications. For example, plans exist to use autonomous vehicles in law enforcement, where UAVs may be equipped not only with cameras and scientific instruments for surveillance and information gathering, but also with weapons, such as rubber bullets, Tasers, and tear gas. These non-military systems, e.g., for law enforcement and public safety, will be natural transitions from the military's decision systems.

How to design and build high assurance human-centric decision systems is largely unknown: while significant research has been published in areas such as intelligent agents, cognitive science, and formal methods, how to relate and integrate the design principles and disparate models in each area is unclear. In combining the research results, difficult questions arise, e.g., what pair-wise model interactions are beneficial? Can design principles in one area be combined with those in another? Designing and building high assurance human-centric decision systems also poses major challenges. Because the human user of these systems performs many complex tasks, he/she will at times become overloaded. A major challenge is how to address human overload. A second major challenge is how to obtain high assurance that these systems behave as intended.

A promising approach to human overload is to use AI (artificial intelligence) methods—in particular, a cognitive model and an agent. The cognitive model's role is to predict human overload, while the agent's role is, upon notification by the cognitive model of human overload, to alert the human or to take control of one or more of the human's tasks. This system design raises major questions. For example, how to design the autonomy—i.e., which tasks to assign to humans and which to the "system," when to switch from human to system control and vice versa, etc.—is unclear. A promising approach to the high assurance problem is to apply software engineering methods—namely, formal modeling and analysis. However, a huge problem is how to obtain the formal system requirements model. Difficult to obtain in general, formal models of requirements are especially hard to obtain for human-centric decision systems given their complexity. Moreover, even if the problem of obtaining a formal requirements model is overcome, major questions remain. For example, how does the formal model represent the requirements of a system composed of a cognitive model and an agent?

To illustrate the process and techniques described in this paper, Section II introduces an example of a human-centric decision system controlling a team of UAVs. To address the research challenges, Section III proposes a novel development process for high assurance human-centric decision systems. In this process, a prototype system which includes a cognitive model and an agent is built, the system requirements are derived from the prototype and expressed as scenarios, a formal system model is synthesized from the scenarios, and ultimately a system is implemented based on the formal model. Section IV describes the results of our research to support this process: a technique for synthesizing formal models from scenarios, a dynamic operator overload model for predicting overload, and a technique for synthesizing user models from human participant studies data.

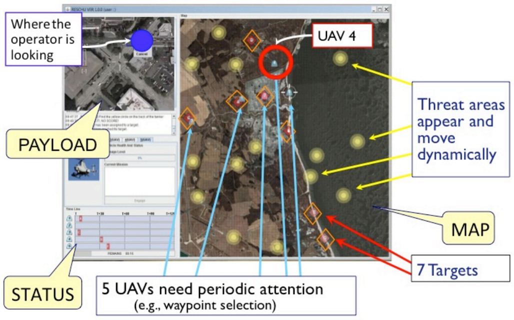
Where the operator is looking
UAV 4
Threat areas appear and move dynamically
PAYLOAD
MAP
7 Targets
STATUS
5 UAVs need periodic attention
(e.g., waypoint selection)

Fig. 1. Operator display of MIT's RESCHU simulator [3].

II. Human-Centric Decision System: Example

Our studies are being conducted in the context of the Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles (RESCHU) [3], a MIT-developed simulator of a decision system in which one human operator controls a team of UAVs. In RESCHU, operators assign and move UAVs to specific target areas, reroute UAVs to avoid threats, and order UAVs to engage targets. The operator can alter the path of a UAV towards its target by manipulating waypoints. Simultaneously, operators perform other tasks (e.g., visual acquisition, surveillance) in scenarios involving urban coastal or inland settings. RESCHU's operator interface (see Fig. 1) has three windows: The Map displays UAVs, targets, and threats. The Status window provides information, such as UAV damage from threats, estimated time for UAVs to reach targets, etc. The Payload window displays status information for other mission tasks. As in other human-centric decision systems, a serious problem in RESCHU is *operator overload*, which occurs when the operator has too many concurrent demands and is unable to handle all in a timely manner.

III. System Development Process

A promising approach to achieving high assurance for human-centric decision systems is Model-Based Development (MBD). In MBD, one or more models of the required system behavior are built, validated (e.g., via simulation) to capture the intended behavior, verified to satisfy required properties, and ultimately used to build the system implementation. Model properties to be verified include *completeness* (no missing cases), *consistency* (no non-determinism), and application properties, for example, safety properties.

While the use of MBD in software practice is growing, a major problem is the lack of good formal requirements models. In many cases, system and software requirements models do not exist at all. Even when they exist, these models are usually expressed ambiguously in languages without an explicit semantics *and* at a low level of abstraction. Ambiguity makes the models hard to analyze formally while the low level of abstraction leads to unneeded implementation bias and also makes the models hard to understand, validate, and change.

To address these problems, researchers have introduced techniques (see, e.g., [27]) for synthesizing formal models from scenarios. Informally, scenarios describe how the system interacts with humans and the system environment to provide the required system services. Because many practitioners already use scenarios to elicit and define requirements, synthesizing formal models from scenarios is highly promising.

Fig. 2 shows a four-step process, an extension of a process introduced in [1], for developing high assurance human-centric decision systems which uses MBD. This is an idealization of the actual process which has more iteration and feedback and may not always proceed in a top-down fashion. In step 1, a prototype is built of the system's conceptual behavior. As Fig. 2 shows, a human operator of the system interacts with a visual display to perform a set of tasks. Because the tasks are complex, an agent is available to assist an overloaded operator and a cognitive model available to predict operator overload.

In step 2, a requirements engineer elicits information about the system requirements from the prototype, determines the *system modes* (externally visible abstractions of the system state), and expresses the requirements as a set of scenarios and system modes. Informally, the system will behave differently in different modes; e.g., if the system is in mode A when a new input arrives, it may respond differently than it would in mode B. In the scenarios, implementation bias, e.g., in RESCHU, how the display represents an endangered UAV, can be avoided by using appropriate abstractions. The requirements engineer also formulates and expresses in precise natural language the required system properties. For RESCHU, an example property is "If endangered (i.e., too close to a threat), then a UAV is under operator control or agent control."

In step 3, the scenarios and systems modes are automatically synthesized into a formal system model, and the model is checked for completeness and consistency and validated (e.g., via simulation) to capture the intended behavior. Moreover, required system properties, such as the example property above, are translated into logical formulae, and the formal model is verified to satisfy these properties, using an appropriate verification tool such as a model checker. Finally, in step 4, the model, along with information such as the platform characteristics, the characteristic and interfaces of I/O devices, etc., is the basis for developing source code, some generated automatically and other code developed manually.

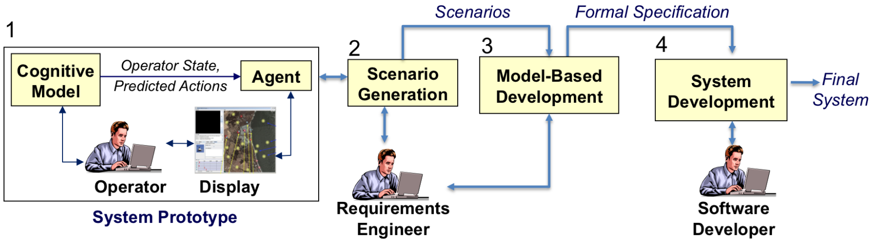

Fig. 2. Development process for a high assurance human-centric decision system.

IV. Building Decision Systems: New Techniques

This section describes new software engineering and AI techniques which support the system development process introduced in Section III. In developing and evaluating these new techniques, we used RESCHU to build a prototype system which relies on a cognitive model to predict operator overload. In future work, an agent will be added to the prototype to assist the operator in performing the assigned tasks.

Section IV-A introduces our new technique for synthesizing formal system requirements models from scenarios and system modes. Sections IV-B and IV-C describe new techniques for overcoming two major problems in the design of human-centric decision systems: how to predict operator overload and how to evaluate agent designs. To address the first problem, Section IV-B introduces the dynamic operator overload model. To address the second problem, Section IV-C describes a new technique for synthesizing user models from data collected in earlier human participant studies.

We expect the technique described in Section IV-A for synthesizing formal system models from scenarios to support steps 2–4 of our development process: Code generated from a validated, verified formal system model should provide high assurance that the system implementation satisfies its requirements. The techniques for predicting operator overload and evaluating agents described in Sections IV-B and IV-C are expected to lead to good designs of human-centric decision systems and will therefore prove useful in developing the prototype system called for in step 1. Moreover, we expect the synthesized user models will also be useful in step 2, e.g., for identifying assumptions about the operator's behavior, and in step 3 as input user data useful for validating and formally verifying the formal model synthesized in step 2.

A. *Formal Modeling and Analysis*

A popular notation for specifying scenarios is that of Message Sequence Charts (MSCs) [16]. Many techniques for synthesizing formal models from MSCs have been proposed but, unlike ours, most (see, e.g., [19], [27], [15]) translate MSCs into software design models rather than system requirements models. Like us, Damas et al. [9] synthesize formal requirements models and have techniques for detecting model incompleteness and generating invariants. Unlike us, however, they use negative MSCs to prohibit certain system behaviors, and, unlike ours, their approach does not use *system modes*.

Formally, a *mode class* is a set of system modes which partition the system's state space [14]. Thus each mode is an equivalence class of system states, and a mode class can be treated as a variable whose possible values are the modes. A goal in choosing modes is to partition the state space in a meaningful way. A major benefit of modes is that modes make the requirements model more concise and thus easier to understand. A related benefit is that modes provide an abstract and intuitive structure for organizing the requirements model. This is especially important for human-centric decision systems like RESCHU, whose behavior is highly complex and whose requirements model will be very large.

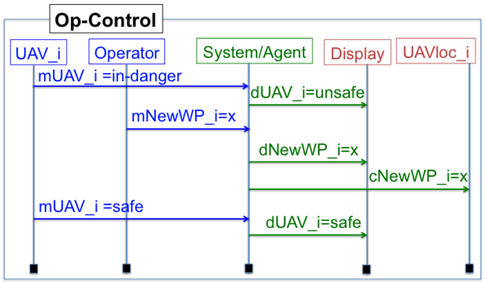

Fig. 3. MSC for operator movement of a UAV's way point.

In our approach, a MSC describes a sequence of stimulus-response behaviors of a state machine model with a next-state function $T : s \times e \rightarrow s'$, where s is the current system state, e a single system input (the stimulus), and s' the new state where the value of system outputs may change (the response).

To illustrate our approach, we consider UAV4 (see the top right corner of Fig. 1) which is close to a threat and in danger. To prevent UAV4 from being damaged or destroyed, the operator, or agent if the operator is overloaded, can insert a waypoint in the UAV's trajectory to avoid the threat.

Fig. 3 contains a MSC specifying the required system behavior when a UAV nears a threat. The label on the outer rectangle contains the MSC's name (Op-Control); the internal rectangles (e.g., Operator, Display) identify the MSC's components, the sources and destinations of the system's inputs and outputs. In Fig. 3, the components on the left are sources of system inputs; the central component represents the system, which maps an input in the current state to zero or more outputs in the new state; and the components on the right produce the system outputs. A prefix m, d, or c on an input or output variable indicates respectively that the variable represents a quantity that is monitored (an input), displayed, or controlled (an output). For example, the variable dUAV_i represents the status of UAV_i which appears on the operator display. Shown on the horizontal lines of the MSC are inputs sent by each source (e.g., mUAV_i=in-danger) and outputs received by each destination (e.g., dUAV_i=unsafe).

The MSC in Fig. 3 contains three stimulus-response behaviors, i.e., (input,output-set) pairs. The first input indicates that UAV_i is in-danger. The system responds by marking UAV_i as unsafe on the display (e.g., circling it in red). Noting from the display that UAV_i is in danger, the operator requests the insertion of a new waypoint x in UAV_i's path to avoid the threat. In response, the system updates the representation of UAV_i's path on the display *and* inserts a new waypoint x which allows the actual UAV_i to bypass the threat. In a MSC, all outputs that follow input i but precede input j must be sent when input i arrives. In our approach, the order of these outputs in the MSC does not matter; all outputs associated with an input are scheduled simultaneously.

In the MSC in Fig. 4, the cognitive model predicts that the operator is distracted, alerts the agent, which then inserts a waypoint to move the UAV to safety. Because the agent is expected to be fully integrated with the system, in this MSC, the agent's behavior and the system behavior are combined.

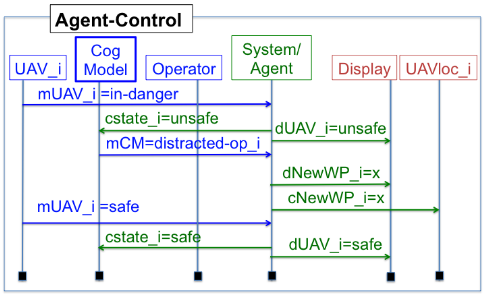

Fig. 4. MSC for agent movement of a UAV's way point.

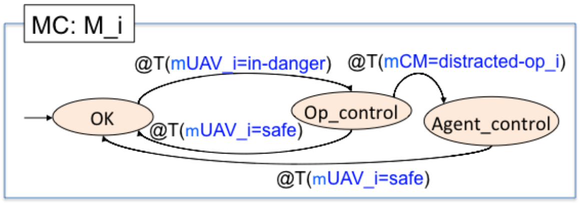

Fig. 5. Mode class diagram showing modes, mode transitions, and inputs.

The mode class diagram in Fig. 5 describes a mode class named M_i containing three modes, each specifying the current system state relative to UAV_i. The transitions shown in Fig. 5 are labeled with the input events that trigger them. For example, the input event @T(UAV_i = in-danger) labeling the transition from OK to Op-control occurs when UAV_i = in-danger becomes *true* after being *false* in the state in which M_i equals OK. According to the table, if UAV_i is safe (far from a threat), M_i equals OK; if UAV_i is in-danger, then M_i equals Op_control when the operator is in control, and Agent_control when the agent is in control.

Our MSC language is defined by the sound formal semantics which underlies the SCR (Software Cost Reduction) requirements notation [14]. We have also developed a sound synthesis algorithm which translates scenarios and mode class diagrams into a formal state machine model [11]. Our synthesis algorithm 1) identifies input and output variables from the MSCs; 2) based on the mode class diagrams, identifies each mode class and its modes, and defines the new mode as a function of an input and the old mode; and 3) from the MSCs and the appropriate mode class, defines the value of each output as a function of an input and the old mode. Currently, the requirements engineer must supplement the synthesized formal model with additional information, e.g., type definitions, the initial state (e.g., the initial mode *OK*), etc.

Table I defines the current state of UAV_i represented on the operator display. This definition is part of a formal model synthesized from the MSCs and the mode transitions in Figures 3-5 using our synthesis algorithm [11]. The table states that if mode M_i equals OK, then dUAV_i=safe, whereas if M_i equals either Op-control or Agent-control, then dUAV_i=unsafe (e.g., circled in red on the operator display).

An advantage of our approach is that the synthesized requirements models can be analyzed using the SCR tools—validated using simulation [12], analyzed automatically for consistency and completeness [14], and verified formally using model checking [13], theorem proving [12], composition [17], and automatically generated invariants [21].

TABLE I
TABLE SYNTHESIZED FROM MSCS AND MODES.

Controlled Variable	Condition	

	M_i=OK	M_i=Op_Control OR
		M_i=Agent_Control

dUAV_i =	safe	unsafe

Formal analysis of a requirements model for a complex system such as RESCHU can expose many classes of requirements errors: Simulation of the formal model may uncover both missing assumptions and missing system behavior. For example, in addition to being safe or in danger, a UAV may be damaged or destroyed; the required system behavior in these cases needs to be specified. Completeness checking may detect missing requirements. For example, what is the required system behavior when neither the operator nor the agent has time (or is available) to move an endangered UAV to avoid a threat? Consistency checking may uncover unwanted non-determinism. As an example, in some cases, the system may assign control of a UAV to both the operator and the agent, clearly an error that needs correction. Finally, the formal model needs to satisfy safety and other critical properties. An example of a required safety property is that the system never allows two UAVs to be too close to one another.

Our plans are to elicit further requirements from RESCHU and to express them as scenarios and mode classes. This will help us evaluate the scalability of our approach We also plan to explore the "feature interaction" problem that arises in defining more than one mode class. If n mode classes are defined, then at any given time the system is in exactly n modes, one from each mode class. To illustrate this, in addition to the mode class M_i, consider another mode class, $\mathtt{Nav}_i = \{\mathtt{TakeOff}, \mathtt{Landing}, \mathtt{Automatic}, \ldots\}$, describing UAV$_i$'s navigation mode. When a UAV is taking off or landing, being near a threat may be impossible. Hence, the requirements model may include the constraint $\mathtt{Nav}_i$ in $\{\mathtt{TakeOff}, \mathtt{Landing}\} \rightarrow M_i = \mathtt{OK}$. We also plan to investigate techniques such as [26] which, in addition to scenarios, uses system properties to refine and extend the requirements model.

B. Cognitive Model

While the purpose of the software engineering technique described above is to help obtain high assurance, the goal of our cognitive modeling research is to develop a model that can predict when a human operator performing a set of tasks will become overloaded. Such a model can help identify situations in which an agent could assist the operator by taking over one or more operator tasks.

As robots have become increasingly autonomous, one human sometimes supervises multiple robots. This single-human-multiple-robot (SHMR) paradigm, while labor-saving, raises human factors concerns about the capabilities and limitations of the human operator who is multitasking in this situation.

Crandall, Cummings, and Mitchell [7], [8] have introduced "fan-out" models to estimate the maximum number of robots a single operator can supervise in a given SHMR context. These estimates are based on *neglect time* (NT), the time a robot may be neglected before its performance falls below a predetermined threshold; *interaction time* (IT), the time a operator needs to interact with a robot to restore its performance to an above threshold level; *wait time attention* (WTA), the time required for the operator to notice that the robot requires maintenance; and *wait time queue* (WTQ), the delay in interacting with the robot when other robots require maintenance at the same time. Fan-out models have been shown to predict the overall performance of operators on SHMR platforms with different numbers of robots to supervise.

One limitation of fan-out models is that they do not predict performance during the course of the SHMR session. Even when the number of robots supervised is within the constraints specified by a fan-out model, there will be times when the operator is overloaded and as a result subject to error. This becomes clear when we consider that the components of the fan-out model (IT, NT, WTA, WTQ) fluctuate from their typical values during the course of a session and at times will conspire to increase the likelihood of error, whether through the increase in the time needed to maintain a vehicle (IT) or the increase in the wait times, WTA and WTQ, as a result of several vehicles requiring attention at the same time.

To address this limitation, we have developed a dynamic model to predict operator overload during the course of a SHMR session. This dynamic operator overload (DOO) model [23], [10], [4] uses predictor variables similar to the fan-out variables. The model was initially developed for the supervision of multiple unmanned air vehicles (UAVs) by one operator. The SHMR platform on which the model was tested is the RESCHU system introduced in Section II.

In our implementation of RESCHU, each UAV is automatically assigned to a target on which to deliver ordnance and automatically launched on a straight-line flight path towards its target. This path appears in the Map window (see Fig. 1). In RESCHU, the operator has two goals: to engage as many vehicles as possible and to prevent damage to the UAVs. Our main concern was preventing UAV damage.

In the DOO model, failure to prevent damage was the indicator of operator overload. In the generation of the model, the unit of analysis was a *path-intersects hazard* (PIH) event, which started from the moment a UAV entered on a collision course with a hazard (i.e., a threat) and ended either when the UAV traversed the hazard, and consequently incurred damage, or else when the operator changed the UAV's trajectory to prevent damage from occurring. Multiple PIH events can overlap in time. The model was created using logistic regression analysis, with NT operationalized as the expected duration of the PIH assuming damage occurs and WTA by the amount of time until the operator first fixates on the relevant hazard. The amount of time required to execute an evasive action on the relevant vehicle (IT) was not included since it is not relevant when damage is avoided. An initial model included WTQ, operationalized as the amount of time spent on actions focused on nonrelevant vehicles (i.e., other than the PIH's vehicle), including hazard evasions or target engagements. While this model was fairly strong, a superior model was generated by replacing WTQ with WTF, the number of eye fixations on objects (UAVs, targets, hazards) not relevant to the PIH event. WTF and WTA are based on eye fixations assessed by an eye tracker. Thus, the predictor variables in the DOO model were NT, WTA, and WTF.

The DOO model provided a strong fit to the data in the baseline experiment from which it was generated. One measure of fit, d', was 2.7, which is quite high. The model was evaluated in a replication of the baseline experiment ($d' = 2.4$). The model's generalizability was assessed by factorial comparisons [25] of the initial RESCHU platform to platform variations, including where engagement was time-constrained and thus harder (d'=2.2), where engagement was automated and thus easier (d'=2.6), where UAVs moved faster (d'=1.7) or slower (d'=2.3) than in the baseline, and where the operator supervised heterogeneous vehicles (UAVs, HALEs,[1] and UUVs) ($d'= 2.4$), rather than only UAVs. In all cases, except the high-speed UAVs, generalization was excellent.

Additionally, the DOO model has been incorporated into the RESCHU platform as the basis for alerting the operator to PIHs as soon as damage is predicted. Here the model assesses the likelihood of damage repeatedly during the course of each PIH event, rather than after the fact, and as soon as it predicts damage is probable, it alerts the user to the threat. The model-based alert system has been tested in several experiments and has been found to reduce instances of damage by as much as half. The model-based alerts typically occur after the elapsing of approximately 20% of the time between identifying a UAV's proximity to a hazard and occurrence of damage (i.e., 20% of NT), thus providing a timely warning.

Thus, the DOO model has practical utility in helping operators cope with overload situations in the course of SHMR supervisory control tasks. It also has theoretical value in highlighting the roles of attention and planning in multitasking contexts. In the future, we will study extensions of our cognitive model. Currently, the cognitive model is focused solely on minimizing damage to UAVs from threats, which may be too limiting for future tasks. Needed is a more comprehensive model, or additional models, that can deal with other tasks (e.g., when to engage a target or where to send a UAV).

C. Agents

In the model-based alert system described above, an alert is issued when the cognitive model detects operator overload. In a more autonomous system, instead of an alert, an agent can be invoked to assist the user in performing tasks. Our goal is to evaluate a large range of agent designs for a given system (e.g., RESCHU) to determine which designs best assist the user. However, evaluating a large number of agent designs using human participants is infeasible given that even small participant studies require substantial time and resources.

[1] High Altitude Long Endurance UAVs.

TABLE II
METHOD FOR SYNTHESIZING AND APPLYING USER MODELS

1. Gather traces of human behavior in an initial participant study.
2. Synthesize user models from traces.
   - Extract feature vectors from traces.
   - Construct an expert operator.
   - Reduce capabilities of expert operator to match user feature vectors.
3. Evaluate whether the models accurately emulate humans.
   - Learn a user classifier.
   - Extract traces from the user models.
   - Attempt to fool the classifier with the models' traces.
   - If the model fails to fool the classifier, go to Step 2.
4. Iterate agent design using the user models:
   - Build/modify an agent.
   - Use the models to test the agent.
5. Test agent performance with human participants.

TABLE III
THE HIGHEST WEIGHTED FEATURES AND THEIR DISCRIMINATION SCORES

Feature	**Weight**
average distance to hazard before action	.71
tally: `change goal`	.49
bigram tally: `engage target`→`change goal`	.29
bigram tally: `change goal`→`engage target`	.23
tally: `engage target`	.18
bigram tally: `change goal`→`add WP`	.13
bigram tally: `change goal`→`change goal`	.12
tally: `delete WP`	.12
average distance between UAVs	.12

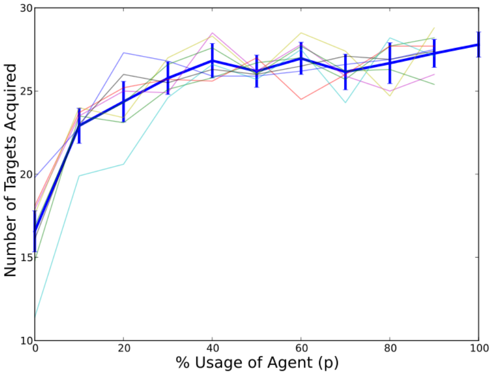

Fig. 6. **User models' performance using agent.** Light lines denote individual user models' performances; bold denotes average performance.

Traditionally, developers evaluate the performance of an agent by conducting human participant studies, iterating the process many times as the agent is refined. The cost and time required for such studies can lead to slow iterations. In contrast, our approach uses synthesized *user models* instead of actual people for some iterations. User models are given the same observations that a human participant would be given and must produce specific actions. Simulated user studies with these models are inexpensive, can be performed quickly, and require no human participant board approval, and thus provide an efficient way to identify problems with an agent design.

Like others, we frame synthesization of user models as an *imitation learning* task [24], where our goal is for the models to learn to operate by imitating human participants. Our work differs from much of the earlier work in imitation learning because, in our case, building an observation-action model without a significant amount of state abstraction is infeasible. The original developers of RESCHU have learned a user model for RESCHU [8], but their model, unlike our user models, is descriptive and cannot be used to generate actual operation in RESCHU. Furthermore, their model does not describe individual users, but an aggregate of users.

Table II provides a high level description of our five-step method for learning user models and using them to evaluate agent designs. Full details can be found in [22]. Our method begins (Step 1) with a set of traces of human behavior (i.e., sequences of observed human actions and environmental updates to the display); in our case study, these traces were obtained from data collected in previous studies with RESCHU involving human participants. The user actions in a trace are captured at a high level of abstraction (e.g., "delete waypoint" instead of "delete the waypoint at location *X* for UAV_*i*"). In Step 2, we synthesize a model for each human user in the data set by first extracting a set of features from each trace for that user and then using the resulting set of feature vectors together with a manually specified *expert operator* (an implementation that behaves like a near optimal user) to produce a model of the user. In our case study, we generated a set of 36 features, including: tallies over each of the high level RESCHU actions, tallies over the ordered pairs or "bigrams" of the high level actions (where the bigram tally `A`→ `B` is the number of times `B` was the first action after `A`), and the average distance between the UAVs. Each feature was assigned a weight, indicating how useful the feature is in distinguishing between users, using a variant of the RELIEF feature weighting algorithm [18]. Table III shows the top weighted features. The expert operator was then modified to reflect the behavior of a user, as described by some of the most heavily weighted features; this modified expert operator is the model for that user.

In Step 3, the behavior of the synthesized user models is compared to that of the actual humans. To do this, a classifier is developed from the original user traces and used to classify traces generated from the synthesized user models (i.e., the classifier identifies which user it thinks produced the trace). When the classifier misidentifies the user, that information is used to improve the synthesized models in the next iteration of Step 2, e.g., by suggesting a different set of features.

Once sufficiently accurate, the user models may be used to evaluate agent designs (Step 4). In our case study, we evaluated an agent, derived from the expert operator, which suggests an action at every step of the test. We learned eight user models and tested their performance in RESCHU with the agent, varying p, the probability that the user model used the agent's suggestion at any time step, from 0 to 100%. Fig. 6 shows the results. When $p = 0$, each model performs as if there is no agent. When $p = 100\%$, the models behave identically to the expert operator. While the increased use of the agent improves performance for all user models, the models with the lowest non-assisted performance benefit the most from the agent. It would be feasible to set $p = 100\%$ because the number of suggestions given by the agent averages 1 every 3.3 seconds (and the agent rarely gives more than one suggestion per second). However, in a human-centric decision system, user participation is essential, and too much automation is inadvisable because it can lead to user disengagement.

Because the user models only approximate human behavior, the performance of the agent design that is finally selected should be validated in a study involving actual human participants (Step 5). Evaluation of the agent design for the RESCHU case study has not yet been performed; it is future work.

Additional future work includes fully automating the process of learning user models from traces, testing the derived models more thoroughly, and integrating agents with the cognitive models described in Section IV-B. Also, we plan to extend our RESCHU case study to include multiple iterations of agent development (i.e., iterating Steps 2-4 of the method).

One major research issue is how to determine the needed level of autonomy for a given system. Too much automation leads to user disengagement; too little results in worsened system performance. Another issue is validation of user models. The hypothesis that a user model matching a user's feature vectors also matches the user's style of operation needs to be validated. The interdependence of the user's features needs to be assessed (e.g., we can omit the "bigram tally: engage `target` → `changegoal`" feature to see if a user model that matches the other features also matches this feature).

V. Related Work

Substantial research has been published on cognitive architectures for control of multi-agent systems (e.g., [6], [20]), and many meetings provide forums for computational cognitive models. Substantial research also exists which applies formal verification to agents and to user interface designs. In [2], Bolton et al. review the large body of research applying formal verification to systems which include "human-automation interaction," including systems containing cognitive models. To our knowledge, however, no current research is applying formal methods, agents, and cognitive modeling to obtain high assurance of the human-centric decision systems introduced in this paper. Although formal verification is included in the model-based development process described in Section III, it is only one component of the process.

VI. Summary and Future Work

This paper has proposed a four-step, model-based process for developing human-centric decision systems, a critical class of systems which will require both high performance and high assurance. To achieve adequate system performance, the process relies on cognitive modeling to predict human behavior and an agent to assist a human in cases of overload. To obtain high assurance, the process relies on formal system modeling and analysis. The paper also introduced two AI techniques, a cognitive model for predicting operator overload and synthesis of user models for evaluating agent designs, and a software engineering technique, synthesis of formal system models from scenarios and system modes, to address the difficult problem of formulating a formal requirements model for a large, complex system. In future work, we will investigate many important technical issues, such as scalability. To what extent can the process and techniques described in the paper deal with human-centric decision systems with a large number of challenging, heterogeneous tasks, a highly dynamic and varied system environment, and sophisticated interaction between a human operator and an intelligent agent?

References

[1] R. Bharadwaj and C. Heitmeyer. Developing high assurance avionics systems with the SCR requirements method. In *Proc. 19th Digital Avionics Sys. Conf.*, 2000.

[2] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu. Using formal verification to evaluate human-automation interaction, a review. Under review.

[3] Y. Boussemart and M. Cummings. Behavioral recognition and prediction of an operator supervising multiple heterogeneous unmanned vehicles. In *Proc., 1st Intern. Conf. on Humans Operating Unmanned Syst.*, 2008.

[4] L. A. Breslow, D. Gartenberg, J. M. McCurry, and J. G. Trafton. Dynamic fan out: predicting real-time overloading of an operator supervising multiple UAVs. Under review.

[5] E. Bumiller and T. Shanker. War evolves with drones, some tiny as bugs. *New York Times*, June 2011.

[6] H.-Q. Chong, A.-H. Tan, and G.-W. Ng. Integrated cognitive architectures: A survey. *Artificial Intelligence Review*, 28:103–130, 2007.

[7] J. W. Crandall, M. A. Goodrich, J. D. R. Olsen, and C. W. Nielsen. Validating human-robot systems in multi-tasking environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(4):438–449, 2005.

[8] M. L. Cummings and P. J. Mitchell. Predicting controller capacity in supervisory control of multiple UAVs. *IEEE Systems, Man, and Cybernetics, Part A: Systems and Humans*, pages 451–460, 2008.

[9] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde. Generating annotated behavior models from end-user scenarios. *IEEE Trans. Software Eng.*, 31(12):1056–1073, 2005.

[10] D. Gartenberg, L. Breslow, J. Park, J. McCurry, and J. Trafton. Adaptive automation and cue invocation: The effect of cue timing on operator error. In *SIGCHI Conf. on Human Factors in Computing Systems*, 2013.

[11] C. Heitmeyer. Synthesizing formal requirements models from modes and message sequence charts. Draft., January 2013.

[12] C. Heitmeyer, M. Archer, R. Bharadwaj, and R. Jeffords. Tools for constructing requirements specifications: The SCR toolset at the age of ten. *Intern. Journal of Computer Systems: Science and Eng.*, 1, 2005.

[13] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Trans. on Softw. Eng.*, 24(11), 1998.

[14] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996.

[15] D. Hirsch, J. Kramer, J. Magee, and S. Uchitel. Modes for software architectures. In *Softw. Architecture, Third Europ. Workshop*, 2006.

[16] ITU. Message sequence charts. Recommendation Z.120, Intern. Telecomm. Union, Telecomm. Standardization Section, 1996.

[17] R. D. Jeffords and C. L. Heitmeyer. A strategy for efficiently verifying requirements. In *ESEC/FSE-11: Proc. 9th Eur. Softw. Eng. Conf./11th ACM SIGSOFT Int. Symp. on Foundations of Softw. Eng.*, 2003.

[18] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proc., 9th Intern. Workshop on Machine Learning*, 1992.

[19] I. Kruger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. In *Intern. Workshop on Distrib. and Parallel Embedded Systems*, 1998.

[20] P. Langley, J. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Sys. Research*, 10(2):141–160, 2009.

[21] E. Leonard, M. Archer, C. Heitmeyer, and R. Jeffords. Direct generation of invariants for reactive models. In *Proc., 10th ACM/IEEE Conf. on Formal Methods and Models for Co-Design (MEMOCODE 2012)*, 2012.

[22] M. Pickett, D. W. Aha, and J. G. Trafton. Acquiring user models to test automated assistants. In *26th Internat. FLAIRS (FLorida AI Society Research Society) Conf.*, May 2013.

[23] R. Ratwani and J. G. Trafton. A real-time eye tracking system for predicting postcompletion errors. *Human Computer Interaction*, 26(3):205–245, 2011.

[24] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In D. H. Sleeman and P. Edwards, editors, *ML*, pages 385–393. Morgan Kaufmann, 1992.

[25] J. A. Swets. *Signal detection theory and ROC analysis in psychology and diagnostics: Collected Papers*. Lawrence Erlbaum Associates, 1996.

[26] S. Uchitel, G. Brunet, and M. Chechik. Behaviour model synthesis from properties and scenarios. In *29th Intern. Conf. on Softw. Eng.*, 2007.

[27] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. on Softw. Eng.*, 29(2), Feb. 2003.