Autom Softw Eng
DOI 10.1007/s10515-014-0157-z

Building high assurance human-centric decision systems

Constance L. Heitmeyer · Marc Pickett · Elizabeth I. Leonard · Myla M. Archer · Indrakshi Ray · David W. Aha · J. Gregory Trafton

Received: 16 October 2013 / Accepted: 16 June 2014

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 is needed that they satisfy their requirements. This paper, the product of an interdisciplinary research team of experts in formal methods, adaptive agents, and cognitive science, addresses this problem by proposing a new process for developing high assurance human-centric decision systems. This process uses AI (artificial intelligence) methods—i.e., a cognitive model to predict human behavior and an adaptive agent to assist the human—to improve system performance, and software engineering methods—i.e., formal modeling and analysis—to obtain high assurance that the system behaves as intended. The paper describes a new method for synthesizing a formal system model from Event Sequence Charts, a variant of Message

C. L. Heitmeyer (✉) · E. I. Leonard · M. M. Archer · D. W. Aha · J. G. Trafton
Naval Research Laboratory, Washington, DC 20375, USA
e-mail: constance.heitmeyer@nrl.navy.mil

E. I. Leonard
e-mail: elizabeth.leonard@nrl.navy.mil

M. M. Archer
e-mail: myla.archer@nrl.navy.mil

D. W. Aha
e-mail: david.aha@nrl.navy.mil

J. G. Trafton
e-mail: greg.trafton@nrl.navy.mil

M. Pickett
Google, Inc., Mountain View, CA 94043, USA
e-mail: marcpickett1@gmail.com

I. Ray
Colorado State University, Fort Collins, CO 80523, USA
e-mail: iray@cs.colostate.edu

Published online: 13 November 2014

Sequence Charts, and a Mode Diagram, a specification of system modes and mode transitions. It also presents results of a new pilot study investigating the optimal level of agent assistance for different users in which the agent design was evaluated using synthesized user models. Finally, it reviews a cognitive model for predicting human overload in complex human-centric systems. To illustrate the development process and our new techniques, we describe a human-centric decision system for controlling unmanned vehicles.

Keywords High assurance · Formal models · Formal methods · Adaptive agents · Cognitive models · Formal model synthesis from scenarios · User model synthesis · User scenarios · System and software requirements

1 Introduction

Many future decision systems will be *human-centric*—i.e., require substantial human oversight and control. One important and growing class of human-centric decision systems are those that manage unmanned vehicles. An estimate is that the U.S. military currently deploys over 7,000 UAVs (Unmanned Air Vehicles) (Bumiller and Shanker 2011). Most of these UAVs, which perform a range of critical tasks including surveillance, reconnaissance, and targeting, are not entirely autonomous but are remotely controlled by humans. Such systems, which are already having a major impact on military applications worldwide, are being used increasingly to extend and complement human capabilities, rather than to replace humans (DSB 2012). Unmanned vehicles are also being deployed increasingly in civilian applications such as law enforcement and public safety. For example, the FBI, U.S. Border Protection Agency, Texas Department of Safety, and U.S. Forest Service, among others, have equipped UAVs with cameras and scientific instruments to conduct surveillance and gather information (Sengupta 2013). The rapid rise in civilian use of unmanned vehicles and recent plans to equip UAVs with nonlethal weapons, e.g., tear gas and pepper spray, have raised serious privacy and other major concerns (US Senate 2013) and led to calls for greater human control and oversight of unmanned vehicles.

Developing human-centric decision systems poses two difficult challenges: (1) how to obtain high assurance that these systems behave as intended, and (2) how to design systems which perform effectively. While a promising approach to (1) is to apply formal modeling and analysis, a huge problem is how to obtain the formal system model: Difficult to obtain in general, formal models of human-centric decision systems are especially hard to obtain given their complexity. A promising approach to (2) is to apply AI (artificial intelligence) methods—i.e., a cognitive model to predict human behavior (e.g., when an operator is overloaded) and an adaptive agent to aid an overloaded operator with his or her assigned tasks. However, how to design these systems raises many difficult questions—e.g., which tasks to assign to humans and which to an agent, when to switch control from a human to an agent and vice versa, and how and when to notify an operator when a task is in urgent need of attention (DSB 2012).

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

This article, which extends a paper presented at the Second International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (Heitmeyer et al. 2013a), describes new research that addresses these challenges. To illustrate the complexity of these systems, Sect. 2.1 presents an example of a human-centric decision system controlling a team of UAVs. To show how high assurance human-centric decision systems can be developed, Sect. 2.2 introduces a system development process in which principles from cognitive science and adaptive agents are used to design the system, scenarios are constructed to specify the required system behavior and a system model synthesized from the scenarios, and, once the model has been validated and verified, a system based on the model is implemented. Sections 3–5 describe the results of our research to support this development process in three areas—formal methods, cognitive science, and adaptive agents.

Section 3, which presents the first of our two research contributions, addresses the formal methods challenge by introducing a new method for synthesizing a formal system model from scenarios. This synthesis method is based on a novel representation of scenarios called a Moded Scenarios Description (MSD), which contains a Mode Diagram and a set of Event Sequence Charts (ESCs), variants of Message Sequence Charts (MSCs) (ITU 1999). To provide a foundation for the method, Sect. 3 also introduces a formal model of a MSD and two algorithms for model synthesis.

Sections 4 and 5 address the system design challenge by introducing principles from cognitive science and adaptive agents useful in designing human-centric decision systems. Section 4, included to illustrate our development process and to provide context for Sections 3 and 5, briefly reviews a cognitive model for predicting operator overload (Breslow et al. 2014), and how the model can be used to determine when to send operator alerts to enhance performance (Gartenberg et al. 2013). After reviewing our ongoing research on synthesizing user models to evaluate agent designs (Pickett et al. 2013), Sect. 5 presents our second research contribution, the results of a new pilot study in which synthesized user models were used to investigate how proactive an adaptive agent should be (i.e., how quickly it should act) to assist a user. This study demonstrates that high-performing user models perform better with moderately proactive agents than with highly proactive agents. In our approach, an adaptive agent would be tightly integrated with a cognitive model.

2 Human-centric decision system: overview

2.1 Example of a human-centric decision system: RESCHU

Our studies are being conducted in the context of the Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles (RESCHU), a MIT-developed simulator of a decision system in which a single human operator controls a team of UAVs (Boussemart and Cummings 2008). 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

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

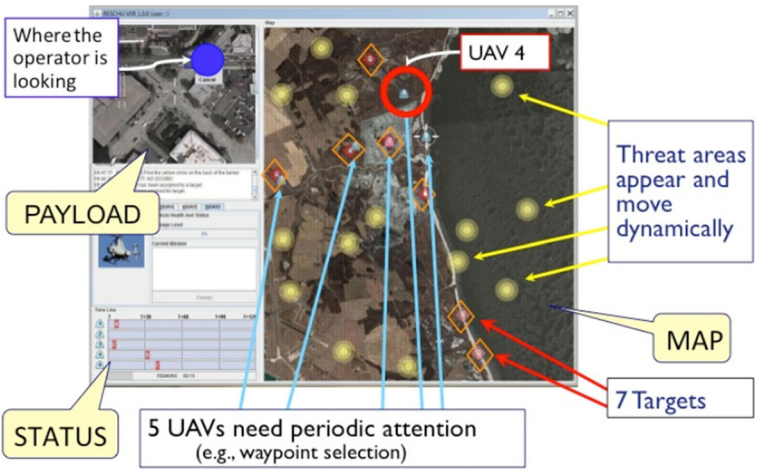

Fig. 1 Operator display of MIT's RESCHU simulator (Boussemart and Cummings 2008)

interface, illustrated in Fig. 1, has three windows. The Map window displays UAVs, targets, and threats (hazards). 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*—the operator has too many concurrent demands and cannot handle all in a timely manner.

2.2 System development process

A promising approach to achieving high assurance for human-centric decision systems is Model-Based Development (MBD) (Heitmeyer et al. 2013b), a variant of Model-Driven Development (MDD) (Selic 2003). 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, such as 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 for synthesizing formal models from scenarios (see, e.g., Whittle and Schumann 2000; Uchitel et al. 2003). 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.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

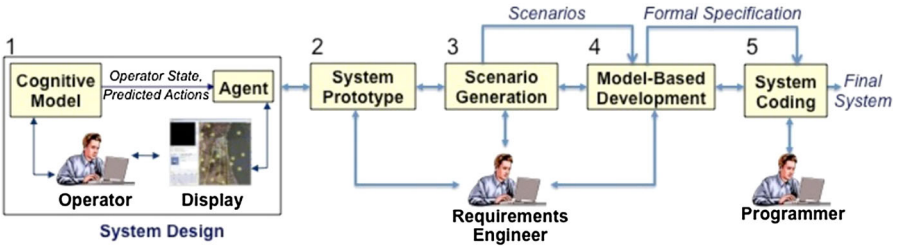

Fig. 2 Development process for a high assurance human-centric decision system

Figure 2 illustrates a five-step process for developing high assurance human-centric decision systems, the extension of a process introduced in Bharadwaj and Heitmeyer (2000). Shown in the figure is an idealization of the actual process, which has more iteration and feedback and which may not always proceed in a top-down fashion. In step 1, principles from cognitive science and adaptive agents are used to formulate the system design. As shown in Fig. 2, in the system, a human operator interacts with a visual display to perform a required set of tasks. Because the tasks are complex, an adaptive agent is available to assist an overloaded operator and a cognitive model to predict operator overload. In step 2, a system prototype is built based on this design. In step 3, 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 in terms of system modes and a set of scenarios. 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 hazard), then a UAV is under operator control or agent control." In step 4, the scenarios and 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 verified to satisfy these properties. Finally, in step 5, the model, along with information such as the platform characteristics, and the characteristics and interfaces of I/O devices, is the basis for developing source code, some code generated automatically from the model using code synthesis techniques such as Leonard and Heitmeyer (2003), Rothamel et al. (2006).

Sections 3–5 describe new software engineering and AI techniques which support this development process. To evaluate these new techniques, we built a prototype system by extending RESCHU with 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. We expect the technique described in Sect. 3 for synthesizing formal system models from scenarios to support steps 3–5 of our development process: Code generated from a validated, verified formal system model should provide high

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

assurance that the system implementation satisfies its requirements. The techniques for predicting operator overload and for evaluating agents described in Sects. 4 and 5 are expected to lead to high quality designs of human-centric decision systems and should therefore prove useful in developing the prototype system called for in steps 1 and 2. Moreover, we expect the synthesized user models described in Sect. 5 will also be useful in step 3, e.g., for identifying assumptions about the operator's behavior, and in step 4 as input user data useful for validating and verifying the formal model synthesized in this step.

3 Synthesis of formal system models from scenarios

3.1 Background: SCR tabular notation, model, and toolset

In 1979–1980, the Software Cost Reduction (SCR) tabular notation was introduced to specify software requirements precisely and unambiguously (Heninger 1980; Alspaugh et al. 1992). This tabular notation has two important benefits. First, software developers find models expressed in table format easy to understand. Second, the tabular notation scales; the large requirements models of practical systems can be concisely represented in tables. In 1996, a formal state machine semantics for models expressed in the SCR tabular notation was presented (Heitmeyer et al. 1996) which includes an important construct of SCR, *system modes* (also called *modes*). Modes provide a system-level abstraction for partitioning the system states into equivalence classes, one class per mode. An important feature of modes is that they are already explicit in many critical software systems (see, e.g., Alspaugh et al. 1992; Heitmeyer and Jeffords 2007). Based on the SCR formal semantics and the notion of modes, a large suite of tools called the SCR toolset has been developed. The toolset includes a consistency checker for detecting well-formedness errors (e.g., type errors, missing cases) in the model specification (Heitmeyer et al. 1996); a simulator for symbolically executing the model to validate that it captures the intended behavior (Heitmeyer et al. 2005); and an invariant generator for automatically generating state invariants from SCR specifications (Jeffords and Heitmeyer 1998; Leonard et al. 2012). Also integrated into the toolset are model checkers (Heitmeyer et al. 1998) for finding violations of desired system properties, such as safety and security properties, and theorem provers (Archer 2001; Jeffords and Heitmeyer 2003; Heitmeyer et al. 2008) for verifying such properties. Tools have also been developed for automatically generating test cases (Gargantini and Heitmeyer 1999) and for synthesizing source code from SCR specifications (Leonard and Heitmeyer 2003; Rothamel et al. 2006).

Although software developers may understand formal models in the tabular format, they have difficulty creating these and other formal models. However, in our experience, when practitioners are presented with a model represented by tables, they can readily understand, modify, and extend the model. The challenge is to produce the initial model. This section describes a Moded Scenarios Description, our solution to this problem. A Moded Scenarios Description contains a set of Event Sequence Charts (ESCs), a Mode Diagram, and a Scenario Constraint. The ESCs look like Message Sequence Charts (ITU 1999), a popular notation used by many practitioners to specify

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

system and software requirements. A Mode Diagram uses system modes to provide a system-level abstraction for combining the ESCs. This section presents an example which shows how selected system requirements for a hazard avoidance task can be specified in terms of a Moded Scenarios Description; and introduces a formal semantics for ESCs and Mode Diagrams, a method for transforming a Moded Scenarios Description into a formal SCR requirements model, and algorithms for computing the update functions of the model.

3.2 Specifying scenarios using Event Sequence Charts and Mode Diagrams

Inspired by Message Sequence Charts (MSCs) (ITU 1999) and their popularity among practitioners, we have developed Event Sequence Charts (ESCs), which have a natural state machine semantics, are easy to change, and are designed to scale better than the traditional basic and hierarchical MSCs. Each ESC contains a set of *entities* and a list of *event sequences*. The entities include the system and a set of environmental entities, the latter consisting of *monitored entities*—entities which the system monitors—and *controlled entities*—entities which the system controls. Each monitored entity is associated with one or more monitored variables and each controlled entity with one or more controlled variables. Each event sequence contains a single *monitored event*, a change in value of a monitored variable, followed by a set, possibly empty, of changes in the values of controlled variables. A monitored event may also cause changes in the values of *term variables*, auxiliary variables designed to make the ESCs more concise and more understandable.

Our approach uses ESCs, a Mode Diagram, and a Scenario Constraint to specify system requirements. A Mode Diagram contains sets of modes and mode transitions. To specify the relationship between ESCs and a Mode Diagram, our approach uses *numeric labels*, positive integers which relate each event sequence in an ESC to the modes and mode transitions in the Mode Diagram. To illustrate our approach, we present two ESCs, a Mode Diagram, and a simple Scenario Constraint specifying (some) required system behavior in a hazard avoidance task. In RESCHU, a UAV's path may cross a hazard area. For an example, see the top of Fig. 1, where UAV 4 is dangerously close to a hazard. To avoid the hazard, either the operator or an agent assisting the operator must modify the UAV's path.

3.2.1 Two examples of Event Sequence Charts

Although ESCs and MSCs look very similar, they have significant differences. While MSCs have been used to specify both system requirements *and* designs, the purpose of ESCs is to specify system and software requirements only. Unlike MSCs which often describe the interactions of many system components, each ESC has only a single system entity and many environmental entities. Further, an event sequence in an ESC includes not only a single monitored event (change in a monitored entity) but all effects of the monitored event, each represented as a change to either a controlled or term variable. Thus a single event in an ESC usually corresponds to a sequence of two or more messages in a MSC. ESCs and MSCs are also semantically different. In

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

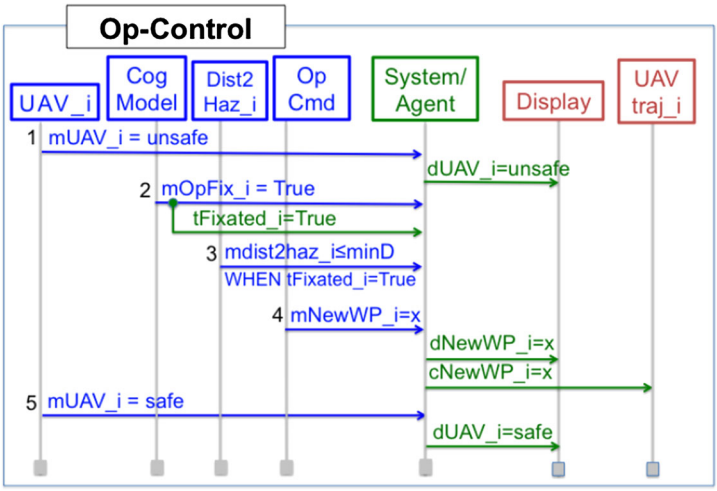

Fig. 3 The ESC named **Op_Control** specifies the sequence of events when the operator adds a waypoint to a UAV's path so the UAV avoids a hazard

an ESC, an event and its immediate effects occur in a single step. In an MSC, an event and its effects may occur in several steps.

Figures 3 and 4 contain two examples of ESCs. These ESCs specify the system's required behavior in the hazard avoidance task for two different sequences of monitored events. The scenario **Op_Control** in Fig. 3 describes the case in which the operator is responsible for hazard avoidance. It contains five event sequences, each assigned a numeric label and interpreted as follows:

1. Upon learning that a UAV is unsafe, the system modifies the display to warn the operator (e.g., by changing the color of the icon which represents the UAV) that the UAV needs attention.
2. The cognitive model learns, e.g., from eye tracker data, that the operator is *fixated* on (paying attention to) the UAV and notifies the system. In response, the system sets the term variable `tFixated_i` to `True`.
3. The system is notified that the UAV is in danger, i.e., the distance between the UAV's location and the hazard is at or below a threshold, the constant `minD`. Because the operator is fixated on the UAV, the system relies on the operator to act to protect the UAV.
4. The operator takes action, i.e., instructs the system to add a new waypoint to the UAV's path to avoid the hazard. In response, the system updates the display to show the UAV's new path once the waypoint is added.
5. The system is notified that the UAV is safe. In response, it updates the display to show that the UAV is no longer in danger.

The scenario **Agent_Control** in Fig. 4 specifies the system requirements when the operator is busy with other tasks, and the agent must protect the UAV from a hazard. The ESC specifying this scenario is interpreted as follows:

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

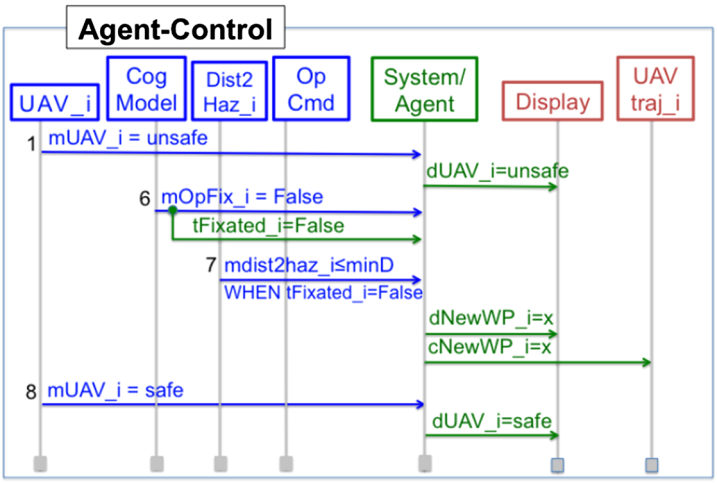

Fig. 4 The ESC **Agent_Control** specifies the sequence of events when the agent adds a waypoint to a UAV's path to avoid a hazard

1. Same behavior as in the **Op_Control** scenario.
6. The cognitive model informs the system that the operator is not fixated on the endangered UAV, causing the system to set `tFixated_i` to `False`.
7. As in the first scenario, the system is notified when the distance between the UAV and a hazard is at or below a distance of `minD`. Because the operator is not fixated on the UAV, the agent must act to protect the UAV by adding a new waypoint to its path. The system then updates the display to show the UAV's new path.
8. Same behavior as event sequence 5 in the **Op_Control** scenario.

3.2.2 Example of a Mode Diagram

The Mode Diagram in Fig. 5 describes a *mode class* called `M_i`; four modes, each a possible value of `M_i`; and five mode transitions. This diagram, together with the ESCs in Figs. 3 and 4 and the Scenario Constraint $\mathcal{C}$ (see Sect. 3.2.3 below), specifies required system behavior in the hazard avoidance task. As shown in Fig. 5, in a Mode Diagram, both transitions and modes have *numeric labels*. Each transition has exactly one numeric label. This label identifies the event sequence containing the monitored event which triggered the transition. For example, in Fig. 5, the transition labeled 1 from mode `OK` to mode `Haz_on_Path` is triggered by the unconditioned monitored event "`mUAV_i = unsafe`" in event sequence 1. This event sequence appears in the ESCs in both Figs. 3 and 4. In Fig. 5, the transition labeled 7 from mode `Haz_on_Path` to mode `Agent_incontrol` is triggered by the conditioned monitored event in event sequence 7 in Fig. 4, namely, "`mdist2haz` $\leq$ `minD WHEN tFixated_i=False`." In a Mode Diagram, each mode has an associated set, perhaps empty, of numeric labels. These numbers, separated by commas, indicate that one or more monitored events may occur in the mode but none triggers a mode transition. For example, in Fig. 5, the "2, 6" label on the mode `Haz_on_Path` refers to the corresponding monitored events in event sequences 2 and 6 in Figs. 3 and 4, respectively.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

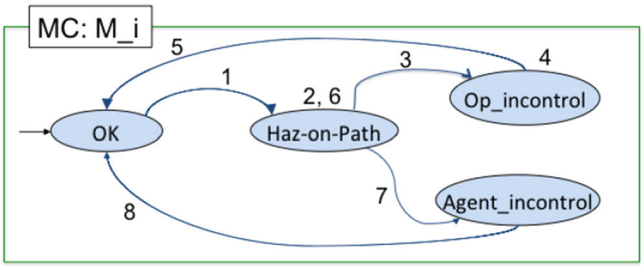

Fig. 5 Mode diagram showing the four modes and mode transitions of mode class M_i, and the numeric labels of monitored events that trigger transitions or can occur in a mode

3.2.3 Scenario Constraint: examples

A Scenario Constraint can define constants, for example, the value of `minD` in the hazard avoidance task. It can also restrict or add to the required system behavior specified in the ESCs and the Mode Diagram. An example of a Scenario Constraint that restricts the system behavior is

$$\texttt{mOpFix\_i=True} \Leftrightarrow \texttt{[tFixated\_i=True]AND[INIT(mOpFix\_i=False)]},$$

where `INIT(_ = _)` is a predicate that constrains the initial value of a variable specified in one or more of the ESCs. This constraint states that term variable `tFixated_i` always has the same value as monitored variable `mOpFix_i` and that the initial value of `mOpFix_i` is `False`. An example of a Scenario Constraint which adds required system behavior is

$$\begin{aligned}&\texttt{M\_i = Op\_incontrol AND mOpFix\_i = True AND mOpFix\_i}' \texttt{= False}\\ &\qquad \rightarrow \texttt{M\_i}' \texttt{= Agent\_incontrol},\end{aligned}$$

where an unprimed variable represents a variable's current value and a primed variable its new value when a monitored event occurs. This constraint states that if the operator stops paying attention when the UAV is close to a hazard, the system switches responsibility for moving the UAV to the agent. This example shows how a constraint may change the behavior specified in the ESCs and the Mode Diagram.

3.3 Formal model of a Moded Scenarios Description

This section formally defines the notions of entities, types, scenario state variables, events, and conditions. These are used to define a set of ESCs, a Mode Diagram, scenario states, and a Scenario Constraint. The ESCs, Mode Diagram, and Scenario Constraint are used in turn to define a *Moded Scenarios Description*. In addition, a set of *numeric labels* are defined that link the event sequences in the ESCs with the modes and mode transitions in the Mode Diagram. Table 1 defines the notation used in this section and Sect. 3.4.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 1 Notation used in this section

Symbol	Usage
B	The Booleans
E^c	A set of controlled events
E^t	A set of term events
F	A map of variables to associated entities
H, K	A set of numeric labels
S	A set of states
V	A map from variables to values
X^m	A monitored event expression
Y	An event sequence chart
Z	The set of integers
Z^+	The set of positive integers
a	A scenario state variable
c	A controlled event
d	A condition
h, k	A numeric label
m	A monitored event
mc	A mode class; a special variable whose set of possible values is a set of modes
op	A binary relational operator such as $=$ or $\leq$
r	A scenario state variable
s	A state; equivalently, a map from state variables to values
$\hat{s}$	A scenario state; equivalently, a map from scenario state variables and mode class to values
t	A term event
EN	A set of entities
MD	A Mode Diagram
RF^{sc}	The set of scenario state variables
RF	The set of state variables
RV	A set of variable/value pairs
TS	A union of type sets; possible values of scenario state variables
TY	The map of variables to their associated type
VS	TS $\cup$ $\mathcal{M}$; Possible values of state variables
$\mathcal{C}$	A Scenario Constraint
$\mathcal{C}^I$	A set of initial value constraints of the form $\texttt{INIT}(r = v)$
$\mathcal{M}$	A set of modes
$\mathcal{M}_T$	A set of mode transitions
$\mathcal{M}_L$	A set of mode/numeric-label-set pairs; in effect, A map from modes to sets of numeric labels
$\mathcal{T}$	A set of values; equivalently, a type set, or a type
$\mathcal{Y}$	A set of Event Sequence Charts
$\mathcal{W}$	A Moded Scenarios Description
Θ	The initial state predicate of a state machine
Σ	A system; a state machine
Υ	An update function
μ	A mode
μ_0	The initial mode
ρ	The transition relation of a state machine

Unless explicitly indicated otherwise, a subscript attached to a symbol associates the object it denotes with an object denoted by the subscript symbol

Journal: **10515** Article No.: **0157** TYPESET DISK LE CP Disp.:**2014/11/8** Pages: **39** Layout: **Small-X**

3.3.1 Event Sequence Charts (ESCs)

We assume the existence of the following sets.

- EN is a set of entities, consisting of a single system and a set of environmental entities. EN partitions into the set EN_M of monitored entities, the set EN_C of controlled entities, and singleton set EN_S containing the system.
- TS is a union of types $\mathcal{T}$, where each type is a nonempty set of values. The set of positive integers $Z^+ = \{1, 2, \ldots\}$ and the set $B = \{\texttt{True}, \texttt{False}\}$ of Boolean truth values are examples of types.
- RF^{sc} is a set of *scenario state variables*, partitioned into set RF_M of monitored variables, set RF_T of terms, and set RF_C of controlled variables. A function TY^{sc} : $\text{RF}^{sc} \rightarrow 2^{\text{TS}}$ maps each variable $r \in \text{RF}^{sc}$ to its set of type correct values $\text{TY}^{sc}(r) \subset$ TS. Another function $F_{\text{EN}} : \text{RF}_M \cup \text{RF}_C \rightarrow \text{EN}$ maps each monitored or controlled variable r to an associated entity x in EN. Thus if $F_{\text{EN}}(r)$ is in EN_M, then r is a monitored variable; if $F_{\text{EN}}(r)$ is in EN_C, then r is a controlled variable.

  Example: In Figs. 3 and 4, the set of entities is EN = {`UAV_i`, `CogModel`, `OpCmd`, `Dist2Haz_i`, `System/Agent`, `Display`, `UAVtraj_i`}, of monitored entities is EN_M = {`UAV_i`, `Dist2Haz_i`, `OpCmd`, `CogModel`}, and of controlled entities is EN_C = {`Display`, `UAVtraj_i`}. Set EN_S = {`System/Agent`} denotes the system. The set RF^{sc} of scenario state variables is {`mUAV_i`, `mOpFix_i`, `mdist2haz_i`, `mNewWP_i`, `tFixated_i`, `dNewWP_i`, `dUAV_i`, `cNewWP_i`}, the set RF_M of monitored variables equals {`mUAV_i`, `mOpFix_i`, `mNewWP_i`, `mdist2haz_i`}, the set RF_T of terms is {`tFixated_i`}, and the set RF_C of controlled variables is {`dNewWP_i`, `cNewWP_i`, `dUAV_i`}. Sample type sets are TY^{sc}(`dUAV_i`) = {`safe`, `unsafe`} and TY^{sc}(`mOpFix_i`) = {`True`, `False`}.

Events and Conditions: *Events* (and *simple conditions*) are triples of the form (r, op, v) where r is a scenario state variable in RF^{sc}, op is a relational operator (=, >, ...), and v is a value in TS. A *monitored event* $m = (r, op, v)$ is an event in which r is in RF_M; a *term event* $t = (r, op, v)$ is an event in which r is in RF_T, and a *controlled event* $c = (r, op, v)$ is an event in which r is in RF_C. In controlled and term events, op always has the value "=".

Example: In event sequence 1 of Fig. 3, the triple $m = (r, op, v)$, where r = `mUAV_i`, op has value "=", and v= `unsafe`, represents the monitored event "`mUAV_i` is `unsafe`." In the `WHEN` clause of event sequence 3 in Fig. 3, the triple $d = (r, op, v)$, where r=`tFixated_i`, op has value "=", and v=`True`, represents the simple condition "`tFixated_i=True`."

A condition d is either `True`, a simple condition, or an expression containing two or more simple conditions joined by the logical operators $\wedge$ and $\vee$ in the standard way. An *unconditioned event* is an event with no associated condition (besides `True`). A *conditioned event* is an event associated with a condition other than `True`.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Example: In Fig. 3, the monitored event "`mUAV_i = unsafe`" in event sequence 1 is an unconditioned monitored event, while the monitored event "`mdist2haz_i` $\leq$ `minD WHEN tFixated_i = True`" in event sequence 3 is a conditioned monitored event.

Event Sequence Charts and Chart Labels: Next, we define an Event Sequence Chart (ESC) Y and an associated set K_Y of numeric labels called *chart labels*.

- An ESC is a sequence Y of event sequences $< y_1, y_2, \ldots, y_n >$. Each event sequence y_i in Y is a triple (i, X_i^m, E_i^c), where i in Z^+ is the *numeric label* of the sequence; X_i^m is a *monitored event expression*; and E_i^c is a set of controlled events. The monitored event expression X_i^m is represented as a triple (m_i, d_i, E_i^t), where m_i is a monitored event, d_i is a condition, and E_i^t is a set of term events triggered by m_i when d_i holds. The sets E_i^t and E_i^c may be empty. In two related event sequence charts Y_1 and Y_2, y_i in Y_1 and y_j in Y_2, $i = j$, implies $X_i^m = X_j^m$ and $E_i^c = E_j^c$. That is, if event sequences in related ESCs have the same numeric label, their monitored event expressions and sets of controlled events are identical.
- Associated with each ESC Y is a set K_Y of *chart labels*. The set $K_Y \subset Z^+$ is defined by $K_Y = \{ i \mid y_i = (i, X_i^m, E_i^c) \in Y\}$.

Example: In Fig. 3, event sequence 1 is denoted as $y_1 = \{1, X_1^m, E_1^c\}$, where $X_1^m = (m_1, d_1, E_1^t)$, m_1 = (`mUAV_i=unsafe`), d_1 = `True`, $E_1^t = \emptyset$, and $E_1^c = \{$(`dUAV_`i, =,`unsafe`)$\}$. Similarly, in the event sequences $y_2 = \{2, X_2^m, E_2^c\}$ and $y_3 = \{3, X_3^m, E_3^c\}$, $X_2^m = (m_2, d_2, E_2^t)$, where m_2 = (`mOpFix_i`, =,`True`), d_2 =`True`, $E_2^t = \{$(`tFixated_i`, =,`True`)$\}$, and $E_2^c = \emptyset$; and $X_3^m = (m_3, d_3, E_3^t)$, where m_3 = (`mdist2haz_i`, $\leq$, `minD`), $d_3 = \{$(`tFixated_i`, =,`True`)$\}$, and $E_3^t = E_3^c = \emptyset$. For the ESC in Fig. 4, the set of chart labels is {1, 6, 7, 8}.

3.3.2 Mode Diagram

A Mode Diagram specifies a simple state machine called a "mode machine". It defines a set of *modes*, an initial mode, and a set of mode transitions. Each transition has exactly one numeric label. Each mode has an associated set, perhaps empty, of numeric labels, each corresponding to a numeric label in an ESC of a monitored event that may occur in that mode. A Mode Diagram MD and its associated set of numeric labels K_{MD} are formally defined next.

- A Mode Diagram is a 5-tuple MD = $(\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L)$, where $\mathcal{M}$ is a set of modes; mc is the *mode class*, a variable with type set $\mathcal{M}$ that names the current mode; $\mu_0 \in \mathcal{M}$ is the initial mode; $\mathcal{M}_T$ is a set of mode transitions, each transition labeled with a unique numeric label; and $\mathcal{M}_L$ is a mapping which associates each mode with a set of numeric labels. The set $\mathcal{M}_T$ of mode transitions is represented as a set of triples (μ_i, k, μ_j), where μ_i and $\mu_j, i \neq j$, are modes in $\mathcal{M}$, and k is the *numeric label* of the transition. The mapping $\mathcal{M}_L$ is represented as a set of ordered pairs (μ_j, H_j), where, for all modes μ_j in $\mathcal{M}$, H_j is the set, possibly empty, of numeric labels associated with mode μ_j. The set $\mathcal{M}_T$ contains information needed

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

to define a mode transition function. The mapping $\mathcal{M}_L$ contains information needed to define controlled variables as functions of modes.

– Associated with a Mode Diagram MD = $(\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L)$ is a set of numeric labels called *diagram labels* K_{MD}, where $K_{MD} \subset Z^+$ is the union $K_{MD} = K_T \cup K_M$ of set K_T of *transition labels* and set K_M of *mode labels*. Set K_T is defined by $K_T = \{k \mid (\mu_i, k, \mu_j) \in \mathcal{M}_T\}$. Set K_M is defined by $K_M = \{h \mid (\mu, H) \in \mathcal{M}_L, h \in H\}$. The sets of transition labels and mode labels cannot overlap; i.e., $K_T \cap K_M = \emptyset$.

Example: In Fig. 5, the set $\mathcal{M}$ of modes is {`OK`, `Haz_on_Path`, `Op_incontrol`, `Agent_incontrol`}, the current mode in mode class mc is `M_i`, and the initial mode μ_0 is `OK`. The set $\mathcal{M}_T$ of mode transitions contains five triples, one for each transition. The transition from `OK` to `Haz_on_Path` is represented by the triple (`OK`, 1, `Haz_on_Path`), where 1 is the transition label. The set of mode-label pairs defining the mapping $\mathcal{M}_L$ is {(`OK`, $\emptyset$), (`Haz_on_Path`, {2, 6}), (`Op_incontrol`, {4}), (`Agent_incontrol`, $\emptyset$)}. The set of transition labels is $K_T = \{1, 3, 5, 7, 8\}$, of mode labels is $K_M = \{2, 4, 6\}$, and of diagram labels is $K_{MD} = \{1, 2, \ldots, 8\}$. As required, $K_T \cap K_M = \emptyset$.

Scenario State: Suppose $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_m\}$ is a set of ESCs and MD is a Mode Diagram MD = $(\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L)$. A *scenario state* $\hat{s}$ is a function $\hat{s} : \mathrm{RF}^{sc} \cup \{mc\} \rightarrow \mathrm{TS} \cup \mathcal{M}$ that maps each scenario state variable in RF^{sc} to a value in its type set $\mathcal{T} \subset \mathrm{TS}$ and the single mode class mc to a mode in $\mathcal{M}$. More precisely, if r is a variable, for all $r \in \mathrm{RF}^{sc} : \hat{s}(r) \in \mathrm{TY}^{sc}(r)$, and for $r = mc : \hat{s}(r) \in \mathcal{M}$. $\mathrm{TY}^{sc}(r) = \mathcal{T}$, where $\mathcal{T} \subset \mathrm{TS}$ is the subset of TS whose members are type-correct values for r. Thus a scenario state is uniquely determined by an assignment of type-correct values to every variable $r \in \mathrm{RF}^{sc} \cup \{mc\}$.

3.3.3 Scenario Constraint

A *Scenario Constraint* $\mathcal{C}$ is a conjunction of one-state and two-state properties, defined, respectively, by predicates on single scenario states and on pairs of scenario states (Heitmeyer et al. 1998). Because each scenario state is represented as a mapping from a set of variables—the scenario state variables and the mode class—to values, useful predicates are typically defined by formulas over the variables of one or two scenario states.[1] A Scenario Constraint $\mathcal{C}$ is *acceptable* if it is satisfiable. We currently restrict conjuncts in $\mathcal{C}$ which use the `INIT` predicate to the form `INIT`$(x = v)$, where x is a scenario state variable. Because $\mathcal{C}$ must be satisfiable, for a given x, there can be at most one value v for which `INIT`$(x = v)$ is a conjunct of $\mathcal{C}$.

[1] An example of a predicate that may be impossible to capture in a formula is the reachability predicate where there are infinitely many possible scenario states.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

3.3.4 Moded Scenarios Description

A Moded Scenarios Description $\mathcal{W}$ is a a triple $\mathcal{W} = (\mathcal{Y}, \text{MD}, C)$, where $\mathcal{Y}$ is a set of ESCs, MD is a Mode Diagram, and $\mathcal{C}$ is a Scenario Constraint. We define the set $K_{\mathcal{Y}}$ of *ESC labels* of $\mathcal{Y}$ as the union of the sets of chart labels associated with the event sequence charts in $\mathcal{Y}$, that is, $K_{\mathcal{Y}} = \cup_{Y \in \mathcal{Y}} K_Y$. If K_{MD} is the set of diagram labels associated with the Mode Diagram MD, then we require $K_{\mathcal{Y}} = K_{MD}$, that is, each numeric label associated with an event sequence in an ESC has a corresponding numeric label associated with a mode or mode transition in MD, and vice versa.

Example: In Figs. 3 and 4, the set of ESC labels is $K_{\mathcal{Y}} = \{1, 2, 3, 4, 5\} \cup \{1, 6, 7, 8\} = \{1, 2, \ldots, 8\}$. In Figs. 3, 4, and 5, $K_{\mathcal{Y}} = K_{MD} = \{1, 2, \ldots, 8\}$ as required.

3.4 Synthesizing a formal model from a Moded Scenarios Description

The objective is to transform a Moded Scenarios Description $\mathcal{W}$, where $\mathcal{W}$ is the triple $(\mathcal{Y}, \text{MD}, \mathcal{C})$, and $\mathcal{Y}$ is a set of ESCs, MD is a Mode Diagram, and $\mathcal{C}$ is a Scenario Constraint, into a formal state machine model. The state machine model is represented as a system $\Sigma = (S, \Theta, \rho)$, where S is a set of states, Θ is an initial state predicate, and ρ is a transform function that maps the current state and a monitored event to a new state. Our method has five steps:

1. *Construct the sets of state variables and values*: Using information in the ESCs and the Mode Diagram, construct the new set RF of state variables and the new set VS of values. Based on these sets, define a new function TY which maps each state variable to its set of possible values. The sets RF and VS and the function TY provide the basis for constructing the system state, conditions, events, and the set S of states in system Σ using the definitions in Heitmeyer et al. (1996).
2. *Construct the initial state predicate*: Based on the definition of the initial mode in the Mode Diagram and information about the initial state in the Scenario Constraint $\mathcal{C}$, define Θ, the initial state predicate.
3. *Construct the system transform*: Based on information in the Mode Diagram MD and the ESCs, construct the transform function ρ. The function ρ is computed using a set of update functions which specify how the values of the dependent variables—the mode class, the controlled variables, and the term variables—change in response to a monitored event.
4. *Simplify and extend the model*: Based on the Scenario Constraint $\mathcal{C}$, modify the update functions.
5. *Analyze and validate the model*: Apply analysis techniques and tools to detect defects in the specification of the model, such as syntax errors and nondeterminism. Once such defects have been removed, other tools can be applied, such as simulators and verifiers to further improve the quality of the model.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

3.4.1 Construct the sets of state variables and values

From the ESCs $\mathcal{Y}$ and the Mode Diagram MD, we construct the following sets.

- $\text{VS} = \text{TS} \cup \mathcal{M}$ is the set of values. VS is the union of the set TS of values defined in the ESCs and the set $\mathcal{M}$ of modes defined in the Mode Diagram MD.
- $\text{RF} = \text{RF}^{sc} \cup \{mc\}$ is the set of state variables. Thus RF is the union of the set RF^{sc} of scenario state variables—the monitored, controlled, and term variables—defined in the ESCs in $\mathcal{Y}$ and the singleton set containing the mode class mc. A new function $\text{TY} : \text{RF} \rightarrow 2^{\text{VS}}$ extends the function TY^{sc}. This function TY maps each variable in RF to its set of type correct values. If the state variable r is a mode class mc, then TY maps r to the set of modes $\mathcal{M}$.

  Example: In Figs. 3, 4, and 5, $\text{TY}(\texttt{dUAV\_i}) = \{\texttt{safe}, \texttt{unsafe}\}$, and $\text{TY}(\texttt{M\_i}) = \{\texttt{OK}, \texttt{Haz\_on\_Path}, \texttt{ Op\_incontrol}, \texttt{Agent\_incontrol}\}$.

Using sets RF and VS and function TY, we define the system state, conditions, events, and the system Σ based on definitions in Heitmeyer et al. (1996).

System State: A *system state* s is a function mapping each state variable r in RF to a value in its type set. More precisely, for all $r \in \text{RF} : s(r) \in \text{TY}(r)$, where $\text{TY}(r) = \mathcal{T}$ is the subset of VS whose members are type-correct values for r.

Conditions: A *condition* is a single-state predicate, i.e., a function with range type Boolean. A *simple condition* is $true$, $false$, or of the form $\tau_1 \odot \tau_2$, where $\odot$ is a relational operator, and τ_1 and τ_2 are any well-formed expressions composed of constants, state variables, and single-state functions in the standard way.

Events: Denoted by "@T", *events* are two-state predicates describing a change in value of at least one variable. A *monitored event*, denoted $@\text{T}(r = v)$, describes the change in monitored variable r in RF_M to value v in $\text{TY}(r)$. A *basic event* is denoted $@\text{T}(c)$, where c is a condition. A *simple conditioned event* has the form $@\text{T}(c)$ WHEN d, where $@\text{T}(c)$ is a basic event and d is a simple condition or a conjunction of simple conditions. It is defined by

$$@\text{T}(c) \text{ WHEN } d \stackrel{\text{def}}{=} \neg c \wedge c' \wedge d,$$

where the unprimed version of c denotes c in the old state, the primed version denotes c in the new state, and the condition d is evaluated in the old state.

System: A system Σ is a triple $\Sigma = (S, \Theta, \rho)$, where S is a set of states, $\Theta \subseteq S$ is the set of initial states, and $\rho \subset S \times S$ is a set of state transitions.

3.4.2 Construct the initial state predicate

Suppose Mode Diagram MD is defined by $\text{MD} = (\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L)$ and the Scenario Constraint $\mathcal{C}$ by $\mathcal{C} = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, where each c_i is a one-

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Algorithm 1: Update Function for a Mode Class

INPUT: Moded Scenarios Description $\mathcal{W} = (\mathcal{Y}, \text{MD} = (\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L), C)$
OUTPUT: Mode Transition Function Υ_{mc}
PRECONDITION: $(K_{\mathcal{Y}} = K_{MD} = (K_T \cup K_M)) \wedge (K_T \cap K_M = \emptyset)$
PRECONDITION: $\forall Y_i, Y_j \in \mathcal{Y} : ((y_i = (i, X_i^m, E_i^c) \in Y_i) \wedge (y_j = (j, X_j^m, E_j^c) \in Y_j) \wedge (i = j)) \implies ((X_i^m = X_j^m) \wedge (E_i^c = E_j^c))$

```
1 Υ_mc = {} ;
2 foreach (μ_i, k, μ_j) ∈ M_T do
3     foreach Y ∈ 𝒴 do
4         if (k, X_k^m = (m_k, d_k, E_k^t), E_k^c) ∈ Y then
5             Υ_mc = Υ_mc ∪ {(μ_i, m_k, d_k, μ_j)};
```

state or two-state predicate. Suppose further that $\Sigma = (S, \Theta, \rho)$. Let $\mathcal{C}^I = \{c_i | c_i \text{ is a conjunct in } \mathcal{C}, c_i = \texttt{INIT}(r = v),\ r \in \text{RF}, v \in \text{VS}\}$ be the set of initial value constraints in $\mathcal{C}$, and let $\text{RV}^I = \{(r, v) | \texttt{INIT}(r = v) \in \mathcal{C}^I\}$. If $s \in S$ is a system state, then $\Theta(s)$ is defined by

$$\Theta(s) = \texttt{True} \Leftrightarrow \forall r \in \text{RF} : \ [r = mc \ \wedge \ s(r) = \mu_0] \ \vee \ [(r, s(r)) \in \text{RV}^I] \ \vee \ [r \neq mc \wedge \ \nexists\, v : (r, v) \in \text{RV}^I]$$

Thus Θ constrains an initial state s to be a state in which the mode class mc has value μ_0, and whenever there is a conjunct of the form $\texttt{INIT}(r = v)$ in $\mathcal{C}$, the value $s(r)$ of r in state s is v. Any variable other than mc whose initial value is not constrained by $\mathcal{C}$ may have any initial value.

3.4.3 Construct the transform function

The transform function ρ is defined as the composition of a set of update functions. There is an update function Υ_r for each dependent (i.e., non-monitored) variable $r \in RF$ specifying how r's value changes from the current state s to the new state s'. To make the definition of the function total, we require that, for all variables r, if r' is not explicitly assigned a value in the new state, then $r' = r$, i.e., in the new state, r's value is unchanged. The update function Υ_r can be represented in the following form:

$$r' = \begin{cases} v_1 & \text{if } E_1 \\ \cdots & \\ v_n & \text{if } E_n \\ r & \text{otherwise} \end{cases} \qquad (*)$$

where each E_i is an expression defined on the variables in states s and s'. We can assume the v_i are distinct because if $v_i = v_j$ for $i \neq j$ then we could replace the two cases "v_i if E_i" and "v_j if E_j" by the single case "v_i if $E_i \vee E_j$".

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

This section presents two algorithms, Algorithm 1 and Algorithm 2, for generating the update functions for mode classes, terms, and controlled variables from the information in a Moded Scenarios Description.

Update functions for mode classes: Algorithm 1 computes the update function Υ_{mc} for mode class mc using information in the Moded Scenarios Description $\mathcal{W}$, where $\mathcal{W} = (\mathcal{Y}, \text{MD}, \mathcal{C})$ and $\text{MD} = (\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L)$. As preconditions, the algorithm requires (1) the set $K_{\mathcal{Y}}$ of chart labels for the set $\mathcal{Y}$ of ESCs equals the set K_{MD} of diagram labels for Mode Diagram MD; (2) set K_{MD} partitions into K_T, the set of transition labels, and K_M, the set of mode labels; and (3) if two ESCs in $\mathcal{Y}$ have event sequences with the same chart label, the event sequences are identical.[2] The algorithm produces a set of 4-tuples (μ_i, m_k, d_k, μ_j), where μ_i and μ_j are modes, m_k is a monitored event, and d_k is a condition. The interpretation of the 4-tuple is that if the event m_k occurs when the system is in mode μ_i and condition d_k is true, then the new mode is μ_j, i.e., $mc' = \mu_j$. To begin, the algorithm initializes Υ_{mc} (line 1). For each mode transition (line 2), the algorithm checks each event sequence in each ESC (line 3) for a chart label that matches the transition label k (line 4). If it finds a matching label, the algorithm adds to Υ_{mc} the transition from mode μ_i to mode μ_j conditioned on the event m_k occurring when d_k holds (line 5).

Because event sequences in related ESCs with the same numeric label are required to have identical monitored, controlled, and term events, Algorithm 1 can be implemented more efficiently: The `for` loop on lines 3–5 can exit once an event sequence with diagram label k is found; not all event sequences with transition label k in all $Y \in \mathcal{Y}$ must be found.

Example: Applying Algorithm 1 to the Mode Diagram for mode class `M_i` and the ESCs in Figs. 3, 4, and 5 generates the following five 4-tuples in set $\Upsilon_{\texttt{M\_i}}$:

$$
\begin{aligned}
\Upsilon_{\texttt{M\_i}} = \{&(\texttt{OK}, (\texttt{mUAV\_i}, =, \texttt{unsafe}), \texttt{True}, \texttt{Haz\_on\_Path}),\\
&(\texttt{Op\_incontrol}, (\texttt{mUAV\_i}, =, \texttt{safe}), \texttt{True}, \texttt{OK}),\\
&(\texttt{Haz\_on\_Path}, (\texttt{mdist2haz\_i}, \leq, \texttt{minD}), (\texttt{tFixated\_i}, =, \texttt{True}),\\
&\quad \texttt{Op\_incontrol}),\\
&(\texttt{Haz\_on\_Path}, (\texttt{mdist2haz\_i}, \leq, \texttt{minD}), (\texttt{tFixated\_i}, =, \texttt{False}),\\
&\quad \texttt{Agent\_incontrol}),\\
&(\texttt{Agent\_incontrol}, (\texttt{mUAV\_i}, =, \texttt{safe}), \texttt{True}, \texttt{OK})\}.
\end{aligned}
$$

From the set $\Upsilon_{\texttt{M\_i}}$ of 4-tuples, the update function for $\texttt{M\_i}'$ can be computed and expressed in the form of $(*)$ as shown below, or equivalently in a special tabular format called in SCR a mode transition table (Heitmeyer et al. 1996). Table 2 contains a mode transition table, a two-state function specifying how the system mode `M_i` changes as a function of the current mode and a new monitored event.

[2] These preconditions ensure that $\mathcal{W}$ is a well-formed Moded Scenarios Description.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Algorithm 2: Update Function for Terms and Controlled Variables

INPUT: Moded Scenarios Description $\mathcal{W} = (\mathcal{Y}, \text{MD} = (\mathcal{M}, mc, \mu_0, \mathcal{M}_T, \mathcal{M}_L), C)$, Controlled Variables RF_C, Term Variables RF_T

OUTPUT: Set of Update Functions $\{\Upsilon_a | a \in \text{RF}_C \cup \text{RF}_T\}$

PRECONDITION: $(K_{\mathcal{Y}} = K_{MD} = (K_T \cup K_M)) \wedge (K_T \cap K_M = \emptyset)$

PRECONDITION: $\forall Y_i, Y_j \in \mathcal{Y} : ((y_i = (i, X_i^m, E_i^c) \in Y_i) \wedge (y_j = (j, X_j^m, E_j^c) \in Y_j) \wedge (i = j)) \implies ((X_i^m = X_j^m) \wedge (E_i^c = E_j^c))$

1 **foreach** $a \in \text{RF}_C \cup \text{RF}_T$ **do**
2 $\Upsilon_a = \{\}$;
3 **foreach** $(\mu_i, k, \mu_j) \in \mathcal{M}_T$ **do**
4 **foreach** $Y \in \mathcal{Y}$ **do**
5 **if** $(k, X_k^m = (m_k, d_k, E_k^t), E_k^c) \in Y$ **then**
6 **foreach** $(r, op, v) \in E_k^c$ **do**
7 $\Upsilon_r = \Upsilon_r \cup \{(m_k, \mu_i, d_k, v)\}$;
8 **foreach** $(r, op, v) \in E_k^t$ **do**
9 $\Upsilon_r = \Upsilon_r \cup \{(m_k, \mu_i, d_k, v)\}$;
10 **foreach** $(\mu_i, H) \in \mathcal{M}_L$ **do**
11 **foreach** $k \in H$ **do**
12 **foreach** $Y \in \mathcal{Y}$ **do**
13 **if** $(k, X_k^m = (m_k, d_k, E_k^t), E_k^c) \in Y$ **then**
14 **foreach** $(r, op, v) \in E_k^c$ **do**
15 $\Upsilon_r = \Upsilon_r \cup \{(m_k, \mu_i, d_k, v)\}$;
16 **foreach** $(r, op, v) \in E_k^t$ **do**
17 $\Upsilon_r = \Upsilon_r \cup \{(m_k, \mu_i, d_k, v)\}$;

$$\texttt{M\_i}' = \begin{cases} \texttt{Haz\_on\_Path} & \text{if } \texttt{M\_i} = \texttt{OK} \wedge \texttt{@T(mUAV\_i} = \texttt{unsafe)} \\ \texttt{Op\_incontrol} & \text{if } \texttt{M\_i} = \texttt{Haz\_on\_Path} \wedge \texttt{@T(mdist2haz\_i} \\ & \leq \texttt{minD)} \wedge \texttt{tFixated\_i} = \texttt{True} \\ \texttt{Agent\_incontrol} & \text{if } \texttt{M\_i} = \texttt{Haz\_on\_Path} \wedge \texttt{@T(mdist2haz\_i} \\ & \leq \texttt{minD)} \wedge \texttt{tFixated\_i} = \texttt{False} \\ \texttt{OK} & \text{if } \texttt{M\_i} \in \{\texttt{Op\_incontrol}, \texttt{Agent\_incontrol}\} \\ & \wedge \texttt{@T(mUAV\_i} = \texttt{safe)} \end{cases}$$

Update functions for terms and controlled variables: Algorithm 2 generates update functions for the term variables and controlled variables. Its inputs are the Moded Scenarios Description $\mathcal{W} = (\mathcal{Y}, \text{MD}, C)$, and the sets of controlled variables RF_C and term variables RF_T. The preconditions are the same as those for Algorithm 1. For each variable r in $\text{RF}_C \cup \text{RF}_T$, the algorithm produces a set of 4-tuples (m_k, μ_k, d_k, v), each representing the assignment $r = v$ when event m_k occurs and $(mc = \mu_k) \wedge d_k$ holds. To begin, the algorithm initializes the Υ_a (lines 1–2). For each mode transition (μ_i, k, μ_j), it then checks each ESC $Y \in \mathcal{Y}$ and if the event sequence indexed by k occurs in Y (line 5), then for each controlled event $(r, op, v) \in E_k^c$ and each term event $(r, op, v) \in E_k^t$, it adds to r's update function an assignment $r = v$ conditioned on

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 2 Mode transition table defining the mode class M_i

Old Mode M_i	Event	New Mode M_i′
OK	@T(mUAV_i = unsafe)	Haz_on_Path
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = True	Op_incontrol
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = False	Agent_incontrol
Op_incontrol, Agent_incontrol	@T(mUAV_i = safe)	OK

Table 3 Event table for tFixated_i

Mode M_i	Event	Event
Haz_on_Path	@T(mOpFix_i = True)	@T(mOpFix_i = False)
tFixated_i′=	True	False

Table 4 Event table for dUAV_i

Mode M_i	Event	Event
OK	@T(mUAV_i= unsafe)	Never
Op_incontrol, Agent_incontrol	Never	@T(mUAV_i= safe)
dUAV_i′ =	unsafe	safe

monitored event m_k and condition $(mc = \mu_i) \wedge d_k$ (lines 7 and 9). For each mode/label-set pair $(\mu_i, H) \in \mathcal{M}_L$, the algorithm checks that each k in H is in some ESC $Y \in \mathcal{Y}$. If so, for each controlled event $(r, op, v) \in E_k^c$ and each term event $(r, op, v) \in E_k^t$, it adds to r's update function an assignment $r = v$ conditioned on monitored event m_k and condition $(mc = \mu_i) \wedge d_k$ (lines 15 and 17). Like Algorithm 1, the `for` loops on lines 4–9 and 12–17 can be exited once an event sequence labeled k is found.

Example: Applying Algorithm 2 to the ESCs and Mode Diagram in Figs. 3, 4, and 5 generates four sets of 4-tuples, one for the term tFixated_i and one each for the controlled variables dUAV_i, dNewWP_i, and cNewWP_i:

- $\Upsilon_{\texttt{tFixated\_i}}$ = {((mOpFix_i, =, true), Haz_on_Path, True, True), ((mOpFix_i, =, false), Haz_on_Path, True, False)}
- $\Upsilon_{\texttt{dUAV\_i}}$ = {((mUAV_i, =, unsafe), OK, True, unsafe), ((mUAV_i, =, safe), Op_incontrol, True, safe), ((mUAV_i, =, safe), Agent_incontrol, True, safe)}
- $\Upsilon_{\texttt{dNewWp\_i}}$ = {((mNewWp_i, =, x), Op_incontrol, True, x), ((mdist2haz_i, ≤, minD), Haz_on_Path, (tFixated_i, =, False), x)}
- $\Upsilon_{\texttt{cNewWp\_i}}$ = {((mNewWp_i, =, x), Op_incontrol, True, x), ((mdist2haz_i, ≤, minD), Haz_on_Path, (tFixated_i, =, False), x)}

The update functions generated by Algorithm 2 for the four dependent variables are presented below. Tables 3, 4, 5, and 6 represent these functions in an alternative tabular format called an SCR event table (Heitmeyer et al. 1996).

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 5 Event table for dNewWp_i

Mode M_i	Event
Op_inControl	@T(mNewWp_i = x)
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = False
dNewWp_i′ =	x

Table 6 Event table for cNewWp_i

Mode M_i	Event
Op_inControl	@T(mNewWp_i = x)
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = False
cNewWp_i′ =	x

$$\texttt{tFixated\_i}' = \begin{cases} \texttt{True} & \text{if } @\texttt{T}(\texttt{mOpFix\_i} = \texttt{True}) \wedge \texttt{M\_i} = \texttt{Haz\_on\_Path} \\ \texttt{False} & \text{if } @\texttt{T}(\texttt{mOpFix\_i} = \texttt{False}) \wedge \texttt{M\_i} = \texttt{Haz\_on\_Path} \\ \texttt{tFixated\_i} & \text{otherwise} \end{cases}$$

$$\texttt{dUAV\_i}' = \begin{cases} \texttt{unsafe} & \text{if } @\texttt{T}(\texttt{mUAV\_i} = \texttt{unsafe}) \wedge \texttt{M\_i} = \texttt{OK} \\ \texttt{safe} & \text{if } @\texttt{T}(\texttt{mUAV\_i} = \texttt{safe}) \\ & \quad \wedge \texttt{M\_i} \in \{\texttt{Op\_incontrol}, \texttt{Agent\_incontrol}\} \\ \texttt{dUAV\_i} & \text{otherwise} \end{cases}$$

$$\texttt{dNewWp\_i}' = \begin{cases} \texttt{x} & \text{if } (@\texttt{T}(\texttt{mNewWp\_i} = \texttt{x}) \wedge \texttt{M\_i} = \texttt{Op\_incontrol}) \vee \\ & \quad (@\texttt{T}(\texttt{mdist2haz\_i} \leq \texttt{minD}) \wedge \texttt{M\_i} = \texttt{Haz\_on\_Path} \\ & \quad \wedge \texttt{tFixated\_i} = \texttt{False}) \\ \texttt{dNewWp\_i} & \text{otherwise} \end{cases}$$

$$\texttt{cNewWp\_i}' = \begin{cases} \texttt{x} & \text{if } (@\texttt{T}(\texttt{mNewWp\_i} = \texttt{x}) \wedge \texttt{M\_i} = \texttt{Op\_incontrol}) \vee \\ & \quad (@\texttt{T}(\texttt{mdist2haz\_i} \leq \texttt{minD}) \wedge \texttt{M\_i} = \texttt{Haz\_on\_Path} \\ & \quad \wedge \texttt{tFixated\_i} = \texttt{False}) \\ \texttt{cNewWp\_i} & \text{otherwise} \end{cases}$$

To describe the semantics of SCR tables, we consider Table 4, a compact representation of the update function $\Upsilon_{\texttt{dUAV\_i}}$ for the controlled variable dUAV_i. The table's last row gives the possible value assignments for dUAV_i, with one column for each possible value. In the table's other rows, the leftmost column contains a value for the mode variable M_i and the remaining columns contain events. The table defines a set of updates to variable dUAV_i as follows. For each row j and column k if M_i has the value in row j and the event in cell (j, k) occurs, then variable dUAV_i is assigned the value in column k of the table's last row. The entry Never in a cell (j, k) indicates that no event can occur when M_i has the value in row j which would assign dUAV_i the value in column k of the last row of the table.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 7 Mode transition table for M_i updated by the Scenario Constraint (see last row)

Old Mode M_i	Event	New Mode M_i′
OK	@T(mUAV_i = unsafe)	Haz_on_Path
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = True	Op_incontrol
Haz_on_Path	@T(mdist2haz_i ≤ minD) WHEN tFixated_i = False	Agent_incontrol
Op_incontrol, Agent_incontrol	@T(mUAV_i = safe)	OK
Op_incontrol	@T(mOpFix_i = False)	Agent_incontrol

Table 8 Condition table for tFixated_i

	Condition	Condition
	mOpFix_i = True	mOpFix_i = False
tFixated_i =	True	False

3.4.4 Simplify and extend the model

As stated in Sect. 3.2.3, a Scenario Constraint $\mathcal{C}$ either restricts or adds to the required system behavior. Conjuncts in $\mathcal{C}$ of the form INIT($r = v$) restrict the system behavior. For example, if INIT(mOpFix_i = False) appears in a conjunct in $\mathcal{C}$, then the initial state predicate Θ contains (mOpFix_i = False). Thus initially the operator in the hazard avoidance task is not fixated on UAV_i.

Consider next the example of a Scenario Constraint from Sect. 3.2.3:

$$\texttt{M\_i} = \texttt{Op\_incontrol} \texttt{AND} \texttt{mOpFix\_i} = \texttt{True} \texttt{AND} \texttt{mOpFix\_i}' = \texttt{False} \rightarrow \texttt{M\_i}' = \texttt{Agent\_incontrol},$$

This constraint adds new required system behavior to the hazard avoidance task specified in Figs. 3, 4, and 5: If a UAV is in danger and the operator becomes distracted, the system passes control to the agent. Clearly, a Scenario Constraint is a more efficient way to add this new behavior than changing the ESCs. While adding a transition to the Mode Diagram (from Op_incontrol to Agent_incontrol) is easy, modifying the ESCs to capture new behavior is problematic because ESCs, like MSCs, are naturally sequential. Moreover, adding a new ESC would repeat most system behavior in Figs. 3 and 4 but add a new event sequence between 3 and 8, call it 9, whereby the agent, notified by the system (still in 9) that the operator is distracted, moves the waypoint and updates the display. In contrast, extending a table to reflect new behavior is trivial. We simply add a new row to Table 2 specifying a transition from Op_incontrol to Agent_incontrol when monitored event mOpFix_i' = False occurs. The last row of Table 7 illustrates this change.

Sometimes, the system behavior specified in a Moded Scenarios Description can be simplified. Consider the term tFixated_i, a temporary variable that records the current value of monitored variable mOpFix_i. The value of tFixated_i may be specified by a simple state invariant: [mOpFix_i = True ⇔ tFixated_i = True]. Thus, rather than the two-state function for tFixated_i defined by Table 3 (an event table), a more abstract one-state function may be defined using an SCR condition table. Table 8 is an SCR condition table, which specifies a (modeless) one-state function defining tFixated_i.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

3.4.5 Analyzing the state machine model

As other researchers have noted (see, e.g., Damas et al. 2009; Whittle and Schumann 2000), models synthesized from scenarios will have defects, e.g., missing initial states, syntax errors, and missing cases. To detect such defects, a tool such as the SCR Consistency Checker (CC) (Heitmeyer et al. 1996, 2005) can be applied. The CC automatically finds these and many other "well-formedness" errors, including undefined, unused, and duplicate variables, circular dependencies, unreachable modes, and unwanted nondeterminism. When it detects a problem, the CC displays the table with the defect and highlights the entry containing the defect. For missing cases and nondeterminism, the CC also presents a counter-example to help the developer diagnose and fix the problem.

To remove defects in the state machine model, a software developer using a Moded Scenarios Description has two choices. First, the developer may fix the state machine model indirectly by making changes to the Moded Scenarios Description from which the state machine model was synthesized, i.e., by modifying the Event Sequence Charts, the Mode Diagram, and/or the Scenario Constraint. For example, the developer may add ESCs, modes, and/or mode transitions to remedy missing cases, or alternatively may remove an ESC to eliminate nondeterminism. Second, the developer may make changes to the tabular specification of the state machine model directly. As noted in Sect. 3.1, in our experience, developers who have difficulty *creating* a tabular specification directly are often able to understand, modify, and extend a tabular specification that is presented to them. Thus, the second alternative is not unrealistic. Further, a practitioner may prefer the second alternative because the effects of changes are more direct and may be easier to predict.

Once the synthesized model is free of well-formedness errors, other tools can be applied—a simulator to ensure that the model captures the intended behavior and verification tools to prove that the model satisfies critical application properties, such as safety and security properties. Eventually, the model may be used as a basis for synthesizing source code, at least for parts of the model, such as the system's control logic and simple functions.

3.5 Model synthesis tool

A prototype tool, implemented in Java, has been developed which implements the first three steps of the synthesis method introduced in Sect. 3.4, thus creating a formal system model. As input, the tool currently accepts a textual representation of a Moded Scenarios Description—i.e., a Mode Diagram, a set of ESCs, and a Scenario Constraint—and constructs (1) a set of type definitions, a mode class and its component modes, and sets of monitored variables, terms, and controlled variables; (2) an initial state predicate; and (3) the system transform function ρ expressed as a set of update functions, one for each mode class, term, and controlled variable. The tool then converts this information into an equivalent SCR specification, expressed in the XML representation used internally by the SCR toolset. This formal model can then be

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

viewed, edited, and analyzed using the SCR toolset, thus providing automated support for steps four and five of our method.

The translation into the formal model is largely straightforward—each set and function that the tool constructs corresponds to a set or function in the SCR model. One limitation is that the event sequence charts generally lack information needed to construct a complete SCR model, and thus the model may not satisfy all typing and consistency requirements. However, the SCR toolset can be used to identify portions of the specification that the user needs to provide. One major source of incompleteness in the generation of the SCR specification is in the variable types; there are cases where it is not possible to infer the type of a variable. For example, consider the event labeled 4 in the Event Sequence Chart shown in Fig. 3. In the formal model synthesized from this chart and from the chart and diagram shown in Figs. 4 and 5, the variables `mNewWP_i`, `cNewWP_i`, and `dNewWP_i` are all assigned the type `UNKTYPE` (i.e., unknown type) in the generated SCR specification; the user will need to replace these definitions with an actual type in order for the SCR specification to type check in the SCR toolset. Other checks performed by the SCR toolset notify the user about variables for which an initial value has not been defined and about non-determinism in function definitions.

3.6 Method and tool validation

Currently, the U.S. Navy and other DoD agencies are exploring systems which use Unmanned Ground Vehicles (UGVs) to assist military personnel in reconnaissance and surveillance, target identification and designation, counter-mine warfare, and chemical, biological, radiological, nuclear or high-yield explosive missions (DSB 2012). Because the UGVs require human oversight and control, these systems are examples of human-centric decision systems. To evaluate the utility of Moded Scenario Descriptions and our model synthesis tool, we conducted a case study of two of these systems. In contrast to RESCHU, these systems use UGVs (rather than UAVs) to perform tasks under operator control, and perform different tasks—cargo loading and explosive ordnance disposal. For each system, a Mode Diagram and ESCs were formulated to specify the required system behavior. To validate our method and tool, we manually translated the ESCs and Mode Diagram for each system into a Moded Scenario Description textual representation and used our prototype synthesis tool to automatically generate a formal model. Domain experts found the ESCs and Mode Diagrams easy to understand and quickly provided feedback on missing and incorrectly specified requirements. Moreover, other than some minor limitations of the tool's parser (e.g., it doesn't accept the label "2–5" on a mode), the translation into a formal model was straightforward.

3.7 Related work

During the last 15 years, there has been a large volume of research published on scenarios and synthesis of formal models from scenarios. This research falls into three categories: (1) Techniques for increasing the expressiveness of scenarios, for making them more formal, and for combining them, both at the same abstraction level and

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

at different abstraction levels; (2) methods for scenario-based synthesis of formal models; and (3) techniques for tool-based analysis of the synthesized models.

Regarding expressiveness, in ESCs, one or more variables are associated with each entity. In contrast, variables are not explicit in MSCs. To address this lack of expressiveness, researchers have combined MSCs with other formalisms that include state variables (see, e.g., Whittle and Schumann 2000; Damas et al. 2005; Uchitel et al. 2009). *Fluents*, propositions expressed by an initial value and sets of initiating and terminating events, are one such formalism (Giannakopoulou and Magee 2003). A fluent may be viewed as a state variable with an initial value and an update function. Fluents can be used to guard the occurrence of an event (Damas et al. 2009), and properties expressed in Fluent Temporal Logic (FTL) can be used to describe preconditions on messages (Uchitel et al. 2009). These uses of fluents are similar to our use of conditioned events. In Damas et al. (2009), fluents are used inside High-level Message Sequence Charts (HMSCs) to indicate which MSCs are enabled and thus determine high level behavior. Used this way, fluents are similar to modes. To constrain UML sequence diagrams (similar to MSCs), Whittle and Schumann use state variables to specify pre- and postconditions on events in the Object Constraint Language (OCL) (Whittle and Schumann 2000). The OCL preconditions play a role similar to our conditions; the OCL postconditions are similar to the effects of our events. Other work on extending MSCs for expressiveness includes specification of negative scenarios (Damas et al. 2005) and the use of properties to specify allowed but not necessarily required system behavior (Uchitel et al. 2009).

Regarding the combination of scenarios, some researchers express relationships between MSCs by attaching *state labels* to a component at appropriate points (see, e.g., Uchitel et al. 2003). Using the same label in multiple scenarios indicates that the component is in the same state in these scenarios and that at this execution point the component's subsequent behavior can be that of any MSC at the point immediately following the label. In contrast, we relate ESCs using the chart labels on the event sequences in the ESCs and the diagram labels in the Mode Diagram. Rather than representing a single identical state (as state labels do in MSCs), modes represent an equivalence class of states, and thus, do *not* indicate points where execution can switch from one ESC to another.

Regarding scenario-based synthesis of formal models, many researchers have synthesized state machine models from MSCs (e.g., Uchitel et al. 2003; Damas et al. 2005, 2009; Uchitel et al. 2009). Uchitel et al. have synthesized Labeled Transition Systems (LTSs) from MSCs using HSMCs and state labels to describe the relationships between the MSCs (Uchitel et al. 2003). Damas et al. have synthesized LTSs from positive and negative scenarios specified as MSCs (Damas et al. 2005). In Damas et al. (2009), an LTS is synthesized from HMSCs with the addition of fluents used as guards on the execution of MSCs in the HMSC's hierarchy. Unfortunately, the state space generated from an LTS specification may be very large. In an industrial case study, for example, 658 LTS states were generated (Uchitel et al. 2003), making the model incomprehensible to the human user. In contrast, our models are expressed as a set of update functions in a tabular format, which leads to a state machine model more easily understood at the local level. Further, modes are a system-level abstraction which leads to fewer states and thus makes the model easy to understand at the global level. In Whittle and

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 9 Notation for the cognitive model

Symbol	Usage
AT	Available time
DMOO	Dynamic model of operator overload
FO	Fan-out
IT	Interaction time
NT	Neglect time
PIH	Path-intersects hazard
WTAA	Wait time for attention allocation
WTQ	Wait time in the decision-making queue
WQF	Wait queue fixations

Schumann (2000), statechart models, rather than LTSs, are synthesized from UML sequence diagrams and OCL constraints, producing a more human-readable model.

Regarding formal model analysis, researchers have proposed many techniques for analyzing the synthesized models. For example, the fluent guards in the synthesized guarded LTS model can be checked for completeness, disjointness, and reachability, and FLT properties can be analyzed by model checking the LTS model (Damas et al. 2009). State invariants that hold for a synthesized LTS model can be generated from the model in combination with either fluents (Damas et al. 2005) or FTL properties (Damas et al. 2009). Alur and Yannakakis (Alur and Yannakakis 1999) describe how individual MSCs, MSC-graphs consisting of multiple MSCs, and HMSCs can be linearized into an automaton and verified using model checking. As stated in Sect. 3.1, the SCR toolset supports all of these techniques, i.e., consistency checking, invariant generation, model checking, and theorem proving, in a single toolset.

4 Cognitive model

In the cognitive sciences, cognitive workload is a notoriously difficult concept to study (Gray and Boehm-Davis 2000). To date, there have been no complete theories of detecting high workload or general methods for predicting when an operator is overloaded. Our approach is to focus our human workload research on a specific problem type—single-human-multiple-robots (SHMR).[3] Recently, we completed a study (Breslow et al. 2014) whose goal was to develop a theoretical model that predicts when a human operator, working in a SHMR environment, becomes overloaded. Such a model can help identify situations in which an adaptive agent could assist the operator by taking over one or more operator tasks. Table 9 defines the notation used in this section.

Crandall et al. proposed that the maximum number of robots that a single human operator could control, called *fan-out* (FO), could be computed as $\mathrm{FO} = \mathrm{NT}/\mathrm{IT} + 1$, where NT (Neglect Time) is the amount of time an operator can ignore a robot before the robot's performance drops below some predetermined level, and IT (Interaction Time) is the amount of time that the operator requires to restore the robot's performance

[3] We treat multiple robots as a generalization of multiple UAVs.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

to the predetermined level (Crandall et al. 2005). This equation defines fan-out as the maximum number of vehicles an operator can interact with (taking time IT for each) while another vehicle is running autonomously (for time NT). The "+1" in the equation accounts for the neglected autonomous vehicle.

While Crandall et al.'s fan-out model focused on task variables, Cummings and Mitchell extended the model to include variables that concerned human information processing (Cummings and Mitchell 2008), specifically those relating to the overhead of delays, or wait times, in addition to the duration of direct interaction (IT) with a vehicle requiring attention. These wait times, WTAA (Wait Time for Attention Allocation) and WTQ (Wait Time in the decision-making Queue), are combined with IT in the denominator of the ratio: $\mathrm{FO} = (\mathrm{NT}/(\mathrm{IT} + \mathrm{WTAA} + \mathrm{WTQ})) + 1$.

One limitation of fan-out models is that they do not predict performance during the course of a 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, WTAA, 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 interaction time needed to maintain a vehicle (IT) or the increase in the wait times, WTAA and WTQ, as a result of several vehicles requiring attention at the same time.

To address this limitation, we have developed a DMOO (dynamic model of operator overload) (Ratwani and Trafton 2011; Gartenberg et al. 2013; Breslow et al. 2014) to predict operator overload during the course of a SHMR session. In applying this model in the case of an operator of multiple UAVs (as in RESCHU), we consider preventing operator overload to be equivalent to preventing UAV damage from a hazard area, because we assume that operator overload is the reason that a UAV takes damage due to a PIH (Path-Intersects Hazard) event. Our model, DMOO, uses three variables—WTAA and two new variables, WQF (Wait Queue Fixations) and AT (Available Time). All of these can be operationalized in a real-time environment:

WTAA: The amount of time to recognize that the focal UAV (i.e., the UAV involved in a PIH event) requires attention. Operationalized as the duration from the start of a PIH event until the relevant hazard was first looked at.

WQF: The number of eye fixations on non-focal objects.

AT: The interval from when a vehicle enters on a collision course with a hazard (i.e., the start of the PIH event) until it would make contact with the hazard if successful evasive action were not taken. This interval is the time available to the operator to recognize and remedy the threat.

Our model of fan-out in Breslow et al. (2014) is expressed in terms of these concepts as $\mathrm{FO} = \mathrm{AT}/(\mathrm{IT} + \mathrm{WTAA} + \mathrm{WTQ})$. Because our DMOO is used in predicting the prevention of damage on a per-event basis, activities during IT—the time spent on actions resulting in the successful avoidance of damage during a PIH event—clearly are not independent of what we are trying to predict. Thus IT is not included in our DMOO logistic regression equation in Breslow et al. (2014):

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

$$\text{Predicted Logit of Damage} = 2.17 + (.00007 * \text{WTAA}) + (.11 * \text{WQF}) - (.00027 * \text{AT})$$

This equation uses WQF rather than WTQ because our experiments measured wait times by observing eye fixations rather than manual actions.

Using an eye-tracker to measure where an operator is looking (called a *fixation*) and how long an operator looks at something (called the *fixation duration*), the experiments recorded operator fixations on a computer screen (Rayner and Morris 1990; Rayner 1998). Different eye movement measures have been shown to be indicators of cognitive processing (Rayner and Morris 1990; Rayner 1998; Just and Carpenter 1976). In the current work, we used eye fixations as a measure of operator attention allocation. For a full justification for this assumption, see Breslow et al. (2014).

As described in Breslow et al. (2014), our DMOO was developed using logistic regression analysis and the results of a RESCHU experiment with 35 participants that recorded the damage outcome and the values of WTAA, WQF, and AT for each PIH event. The model provided a strong fit to the data from which it was generated. One measure of fit, d' (Fawcett 2006; Swets 1996), was 2.7; a value of over 2 is typically acceptable for automation work. The model was later evaluated in a replication of the baseline experiment ($d' = 2.4$). The model's generalizability was assessed by factorial comparisons (Swets 1996) 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—HALEs (High Altitude Long Endurance UAVs), UUVs (Unmanned Underwater Vehicles), and UAVs—($d' = 2.4$), rather than only UAVs. In all cases, except the high-altitude UAVs, generalization was excellent.

Additionally, the DMOO has been incorporated into the RESCHU platform as the basis for alerting the operator to PIHs as soon as a high likelihood of damage was 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 found to reduce instances of damage by as much as half (Breslow et al. 2014; Gartenberg et al. 2013). The model-based alerts typically occur after an elapse of approximately 20 % of the time between identifying a UAV's proximity to a hazard and occurrence of damage (i.e., 20 % of AT), thus providing a timely warning. Thus, the DMOO has practical utility in helping operators cope with overload in SHMR supervisory control tasks. It also has theoretical value in highlighting the roles of attention and planning in multitasking contexts.

5 Adaptive agents

In a human-centric decision system, an adaptive agent can assist the operator when he or she is overloaded. Our goal is to evaluate a large variety of agent designs for a given

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 10 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
Train a user classifier on the actual human traces
Extract traces from the user models
See if classifier is fooled into thinking the models' traces came from the corresponding humans
If the model fails to fool the classifier, go back to Step 2
4. Iterate agent design using the user models:
Build/modify an agent
Use the user models to test the agent
Stop iteration if agent performance meets prespecified performance metric
(E.g., if agent helps users achieve fewer missed targets per session)
5. Test agent performance with human participants

system (e.g., RESCHU) to determine which designs best assist the user. Traditionally, developers evaluate an agent's performance by conducting human participant studies, iterating the process many times as the agent is refined. However, evaluating all agent designs using human participants is problematic given that even small participant studies require substantial time and resources, including any required for institutional approval, resulting in very slow iterations. To address this problem, our approach uses synthesized *user models* instead of humans for some iterations. User models are given the same observations a human would be given and must produce specific actions. Simulated user studies with these models are inexpensive, can be performed quickly, and require no approval process. Hence, they are an efficient way to identify problems with an agent design.

Like others, we frame synthesis of user models as an *imitation learning* task (Sammut et al. 1992), where the goal is for the models to learn to operate by imitating the behavior of humans. Our work differs from most earlier work in imitation learning because, in our case, building an observation-action model without a substantial amount of state abstraction is infeasible. One exception was described by the developers of RESCHU (Boussemart and Cummings 2008), who also used learning to obtain abstract user models for RESCHU. However, unlike ours, their models are descriptive and cannot be used to generate actual operations in RESCHU. Furthermore, their models do not describe individual users, but an aggregate of users.

5.1 Case study

Table 10 summarizes our five-step method for learning user models and applying them to evaluate agent designs. This section describes the first four steps of this method and their initial application involving RESCHU. The method's final step, human evaluation of an agent, was not performed in this study.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Table 11 The highest weighted features and their discrimination scores, where high level actions are shown in `this font`

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

Step 1. Gathering Initial Traces: The method begins 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.

We collected ten traces from each of eight human users. Each trace recorded, twice a second for a 5-min period, the user's actions and observations available to the user while operating RESCHU. The available observations include the locations of UAVs, hazards, and targets. They also include waypoint and target-assignment information, the location of the sequence of waypoints for each UAV, and the target (if any) assigned to each UAV. 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 RESCHU, the user can perform six high level actions. Five are available at any time: change the assignments of UAVs to targets (`change goal`), insert a waypoint (`insert WP`), delete a waypoint (`delete WP`), move a waypoint (`move WP`), and do nothing (`NOP`). The sixth action, engage a target (`engage target`), is available only when a UAV has arrived at its assigned target. Though we retained no identifying information about users, each is referred to by an index from 1 to 8.

Step 2. Synthesizing User Models: In this step, we synthesize a model for each human user in the data set. Because the user may have a large number of possible observations in RESCHU ($\sim 10^{124}$), directly building an observation-action model such as those in Sammut et al. (1992) and Šuc et al. (2004) would require significant state abstraction. Instead, we produced our user models by first extracting a vector of features from each user trace, and then using them with a manually specified *expert operator* (an implementation that behaves like a near-optimal user). In our case study, we identified a set of 36 features, and based on these features extracted a feature vector from each of the 80 traces. We then assigned weights to features using a variant of the RELIEF feature weighting algorithm (Kira and Rendell 1992), which allowed us to judge which features best distinguish our eight users (i.e., features with a high variability among users, but low variability within vectors generated by a single user). We chose RELIEF over other feature weighting algorithms due to its popularity and ease of implementation. Table 11 shows the top weighted features.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

Features considered included tallies over the six high level actions, tallies over the 25 "bigrams" for the five non-NOP actions (where A→ B is the number of times B was the first non-NOP action after A), the average proximity to a hazard a UAV reached before a user rerouted it, the average distance between the UAVs, the average time a UAV was idle (i.e., between when it arrived at its target and when the user engaged that target), the average number of (non-NOP) actions the user took divided by the user's performance score (as a measure of action efficiency), and the average amount of time a user waited between when a UAV's path crossed a hazard area and taking action to reroute the UAV. Note that the `engage target` tally is the same as the performance measure (the number of targets engaged).

We used the results of RELIEF to build models of individual users. To begin, we built a simple rule-based expert operator who followed three rules:

1. **if** there is a UAV waiting at a target, `engage` that target.
2. **else if** the UAV target assignments are "suboptimal" (using a greedy nearest target to UAV calculation), reassign the targets greedily (using `change goals`).
3. **else if** a UAV's flight path crosses a hazard area, reroute the UAV around the hazard (using `add WP`).

This simple expert operator performs well, acquiring an average of 27.3 targets per 5-min session, compared to an average of 21.7 targets acquired per session for human users. Since reaction time is a limiting factor in human-user performance for RESCHU, our expert operator outperformed our best human operator; the highest scoring human user averaged 24.5 targets per session.

This expert operator served as a basis for constructing individual user models. Based on the results shown in Table 11, we constrained our expert operator to mimic the two highest weighted features: the user's average distance to hazard before an evasive action and the user's average total number of `change goal` actions. We also constrained the expert operator to mimic the user's average UAV idle time, a feature that, while not one of the highest weighted features, was easy to implement: The expert operator can simply delay engaging a target to better match the user's average UAV idle time.

Step 3. Evaluating the User Models: In this step, we compared the behavior of the synthesized user models to that of the actual humans. Because there is substantial interdependence among the features in a user's feature vector, we hypothesize that it would be difficult to generate a trace that produces a similar feature vector but operates with a markedly different style. We implemented the sub-steps of Step 3 and trained a classifier to distinguish human users by their feature vectors, extracted traces from the user models, then tested whether the classifier could distinguish human users from their models.

For each of our eight human users, we had ten vectors of 36 real-valued features. Thus, training a classifier was a straightforward application of multiclass supervised learning, with the classes being the eight individual users. We trained a suite of 35 classifiers from the PyMVPA toolkit (Hanke et al. 2009) using leave-one-out cross-validation (i.e., train the classifiers on nine of a user's vectors and use the tenth for

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

validation, repeating this process ten times so that each vector is used for validation once). Included in the classifier suite were various parameterizations of Bayes Logistic Regression, Naive Bayes, Gaussian Process Classifier, k-Nearest Neighbors, Least-Angle Regression, Sparse Multinomial Logistic Regression, Elastic Networks, and Support Vector Machines (SVMs) with various kernels. The most accurate of these classifiers for our data was an SVM with a polynomial kernel (Poly-SVM), which yields an average of 62.5 % accuracy on the test set (compared to 12.5 % accuracy for a random classifier).

Applying the synthesized user models, we generated ten new traces for each of our eight users with the aim of mimicking each user. Poly-SVM's classification accuracy was 27.5 % on these traces. That is, given a trace generated by the model for user u, the classifier correctly identified the trace as coming from user u 27.5 % of the time. (In comparison, a random classifier yields an accuracy of only 12.5 %.) For this classifier, we consider a practical upper limit to be 62.5 % because this is the classifier's accuracy on actual human traces. While our classifier's identification of the user models is significantly better than random, our models do not emulate their human counterparts with sufficient accuracy. In practice, we want both the classifier and the model accuracy to be higher. The current results may be improved by returning to Step 2 and using the classifier's predictions to guide our selections of new combinations of features for use in synthesizing user models.

Step 4. Testing an Agent Design: Once the user models meet a pre-specified level of classification accuracy, they may be applied to evaluate agent designs. In our case study, we implemented an agent that constantly "suggested" an action (i.e., the action the expert operator would perform) at every step of the session. The user (or the user model) then had the choice of executing a high level action, or invoking the agent, which would then execute its suggested action. We had no model for how users would use an agent (as there were no agents available to them when we collected their traces), so we parameterized the user models' interaction with the agent using a parameter p that denotes the percent probability that a user model will invoke the agent at any particular time.

We learned eight user models and tested their performance in RESCHU with the agent, varying p from 0 to 100 %. The performance metric used was the number of targets acquired. Figure 6 shows the results of our experiment. When $p = 0\,\%$, each model performs as if there is no agent. When $p = 100\,\%$, the models behave identically to the expert operator, i.e., as if the behavior is totally automated. While performance increases with p for all user models, the models with the lowest non-assisted performance benefit the most from the agent. For this application, it would be feasible to set $p = 100\,\%$, because the number of suggestions given by the agent averages one every 3.3 s (and the agent rarely provides more than one suggestion per second). However, in a human-centric decision system, user participation is essential, and too much automation (setting p to 100 %) inadvisable because it will lead to user disengagement.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

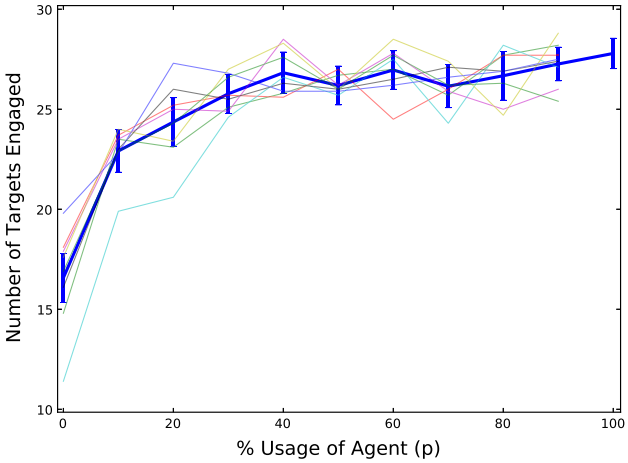

Fig. 6 User models' performance using agent. The *thin lines* denote individual user models' performances, while the *thick line* denotes average performance

5.2 Adjusting agent interaction for different user types

Although the results presented above and described in Pickett et al. (2013) show that assistance from an agent can improve a user's (or user model's) performance, an overly active agent may interfere with a user's goals, causing a decrease in performance. Thus, the question of *when* an agent should provide assistance to a user must also be addressed. This section describes new results in which a modified agent "stalls" individual UAVs when the UAV comes within a certain *threshold* distance t to a hazard. We found that the optimal choice of this threshold varies for different user models, with higher-performing user models generally requiring a shorter distance (and therefore less assistance) than lower-performing users.

These results could be used to design an agent that assists two classes of operator, low-performing (Class 1) and high-performing (Class 2). Figure 7 shows a Mode Diagram and an ESC for such a system, where `min1` is the threshold for stalling a UAV for Class 1 operators and `min2` the threshold for Class 2 operators. This agent design also has a threshold `tooclose`, the point at which an operator cannot act in time to prevent damage to a UAV and the agent must take corrective action. For a Class 1 operator, the agent pauses the UAV when distance `mdist2haz_i` is less than `min1` but not less than `min2`. In Fig. 7, this required system behavior is represented by transition 4 in the Mode Diagram and event sequence 4 in the ESC "OpControl_b". For a Class 2 operator, the system pauses the UAV when `mdist2haz_i` is less than `min2` but greater than `tooclose`. The Mode Diagram also covers cases in which the operator adds a waypoint when `mdist2haz_i` $\geq$ `min1` (thus the agent need not pause the UAV) and when the UAV is too close to the hazard (`mdist2haz_i` $<$ `tooclose`) for the operator to act in time (thus the Agent moves the waypoint). To save space, Fig. 7 shows only one ESC. Note that the Scenario Constraint in a Moded Scenarios Description for this system would include the predicate "`min1` $>$ `min2` $>$ `tooclose`" and define the constants `min1`, `min2`, and `tooclose`.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

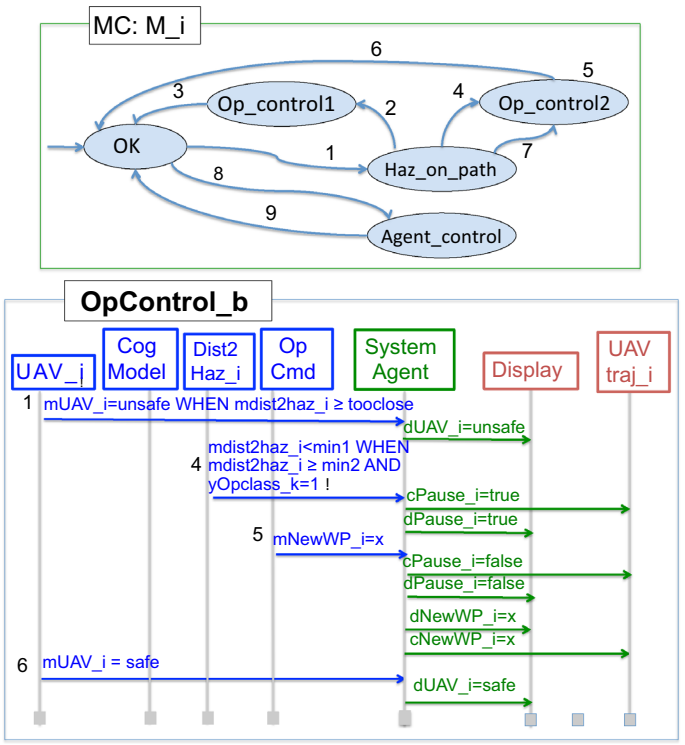

Fig. 7 Mode diagram and Event Sequence Chart for a system with two operator classes

The goal of our experiment was to collect data about how different thresholds affect operator performance, since such data could be used to determine appropriate thresholds for use in designing a threshold-triggered agent. As noted above, in this experiment, when a threshold is crossed, the agent pauses the UAV (i.e., as if it is flying in a tight holding pattern). This modification helps the operator maintain situational awareness; missions cannot be accomplished without the user's participation.

While stalling a UAV prevents it from entering a hazard area and being damaged, it can also delay the UAV's arrival at its target. To find the optimal value for the threshold distance t for each user model, we ran each of our eight user models using agents with t ranging from 0 (the agent never assists) to 40 (the agent frequently assists). Figure 8 shows the performance results (averaged over 100 trials) for the lowest and highest performing user models. Compared to operating without an agent, the higher-performing user model benefits most from a low distance threshold (i.e., a less proactive agent), but has reduced performance when interacting with an overly intrusive agent. We hypothesize that this is because the optimal strategy for a user model might sometimes be to fly a UAV near (but not entering) a hazard area. An overly intrusive agent would prevent this possibility. In contrast, our lower-performing user model's performance did not substantially vary with t.

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

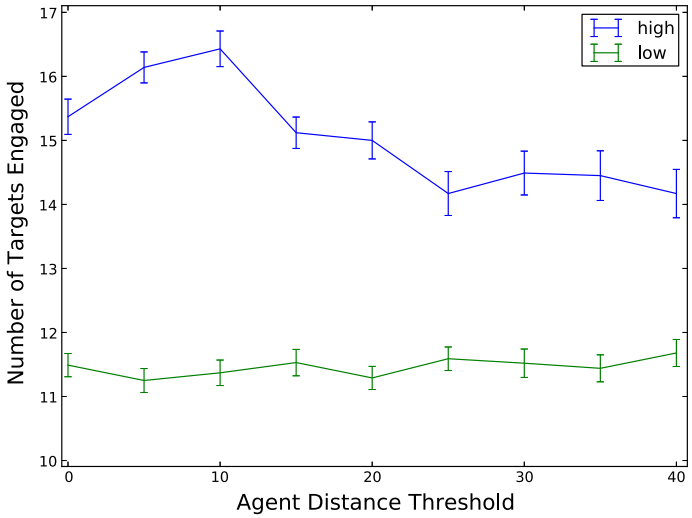

Fig. 8 User models' performance using agent with different target distance thresholds. A high-performing user model requires less help from the agent

The modified agent used in this experiment allows for tight integration with cognitive models, such as those described in Sect. 4. Instead of using the distance from a UAV to a hazard as the threshold for determining when an agent should assist a user, we could instead define a threshold based on the probability that a user will allow a UAV to enter a hazard area. A cognitive model computes this probability.

6 Future work

We are currently developing a graphical front-end for the model synthesis tool that would allow a practitioner to develop graphical representations of ESCs that resemble those shown in Figs. 3 and 4 (see Sect. 3.2.1) and to graphically specify the Mode Diagram for a Moded Scenarios Description (as in Fig. 5 of Sect. 3.2.2). This graphical front-end is being developed using the NetBeans Visual Library, which is already used in the SCR toolset to construct variable dependency graphs.

A future issue is how to synthesize and compose models with more than a single mode class. In the hazard avoidance task, the operator manages a team of many UAVs, rather than a single UAV. By creating n versions of the system model synthesized from the ESCs and Mode Diagram in Figs. 3, 4, and 5, the developer can create n models, each with a mode class `M_`i, i $= 1, 2, \ldots, n$. Composing these will produce a system that, with the agent, protects n UAVs from hazards, rather than a single one. When the individual UAVs in a team are independent of one another, the composition is essentially parallel, and can be done by simply combining the tables of all of the n models. When models of systems that interact with one another are composed, the composition may need to add constraints that limit the interaction. For example, particular modes of two or more systems may be incompatible; e.g., a UAV that is avoiding a hazard should not initiate a targeting task. How best to capture this

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

incompatibility (i.e., "feature interaction") using scenarios and tabular requirements specifications is an open question.

The current cognitive model focuses on the hazard avoidance task, predicting when an operator becomes overloaded and how to minimize damage to UAVs from threats. While the theory is applicable in any task in which the operator manages multiple UAVs, it has not yet been applied to other tasks. Ongoing work is examining how to use our theoretical model, DMOO, to predict operator overload during secondary tasks like engaging a target and also increasing levels of autonomy in the system as the probability of taking damage increases.

Ongoing research also includes fully automating the process of learning user models from traces, and testing the derived models more thoroughly. We plan to extend our RESCHU case study to include many iterations of agent development (i.e., iterating Steps 2–4 of the method). One major research issue is validating user models. The hypothesis that a user model that matches a user's feature vector will also match the user's style of operation requires validation. The interdependence of the user's features also needs to be assessed (e.g., by omitting the "bigram tally: `engage target` → `change goal`" feature to see if a user model matching the other features also matches this feature). Future work on adaptive agents also includes improvements in each of the five steps of our method for learning user models and applying them to evaluate agent designs. In particular, we have left as future work agent design validation using human participants (Step 5). We also plan to iterate Steps 2 and 3 with the aim of improving the fidelity of our user models.

7 Conclusions

This paper has presented a Moded Scenarios Description, a new technique which allows a developer to specify the required behavior of complex systems in terms of scenarios. A Moded Scenarios Description includes Event Sequence Charts, a variant of Message Sequence Charts; a Mode Diagram; and a Scenario Constraint. The paper also introduced a formal model for a Moded Scenarios Description, a method for transforming the description into a formal state machine model, and two algorithms that construct the next-state function of the state machine model from the description. The state machine model is represented in a scalable tabular format which has proven in practice to be both human readable and easy to change. We illustrated our new method by applying it to a hazard avoidance task for a system that manages UAVs, using an adaptive agent to assist the system operator.

After reviewing the process for synthesizing user models presented in Pickett et al. (2013) and Heitmeyer et al. (2013a), this paper introduced the results of new research showing how different user models for a hazard avoidance task perform under different values for the distance threshold t at which the adaptive agent is triggered in a hazard avoidance scenario. This agent design facilitates integrating the agent and a cognitive model of the user; a slight variation of the agent can threshold on predictive values returned by the cognitive model.

The cognitive model can predict in real-time when an operator is overloaded. The model is theoretically based and uses cognitive science and engineering principles to

Journal: 10515 Article No.: 0157 TYPESET DISK LE CP Disp.:2014/11/8 Pages: 39 Layout: Small-X

monitor operator workload. The model has been tested across multiple experiments and has been shown to reduce damage to UAVs (Breslow et al. 2014; Gartenberg et al. 2013). The cognitive model is important not only because it is able to predict operator workload and prevent UAV damage, but also because it can be integrated into other systems where understanding whether an operator is overloaded is important. Integrating the theoretical model of workload, the agent approach to automation, and high assurance methods has the potential to significantly improve the overall performance of systems and to provide high assurance that human-centric decision systems behave as intended.

Acknowledgments We gratefully acknowledge the contributions of Len Breslow to the research on cognitive models, of Carolyn Gasarch of NRL who built the prototype model synthesis tool, and of Michael Thomas of the University of Maryland who applied the synthesis tool to the UGV applications. This research is supported by the Office of Naval Research.

References

Alspaugh, T.A., Faulk, S.R., Britton, K.H., Parker, R.A., Parnas, D.L., Shore, J.E.: Software requirements for the A-7E aircraft. Tech. Rep. NRL-9194, Naval Research Laboratory, Washington, DC (1992)

Alur, R., Yannakakis, M.: Model checking of Message Sequence Charts. Proceedings of the 10th International Conference on Concurrency Theory (CONCUR), pp. 114–129. Eindhoven, The Netherlands (1999)

Archer, M.: TAME: Using PVS strategies for special-purpose theorem proving. Ann Math Artif Intell **29**(1–4), 131–189 (2001)

Bharadwaj, R., Heitmeyer, C.: Developing high assurance avionics systems with the SCR requirements method. In: Proceedings of the 19th Digital Avionics Systems Conference (DASC), Philadelphia, Pennsylvania (2000)

Boussemart, Y., Cummings, M.: Behavioral recognition and prediction of an operator supervising multiple heterogeneous unmanned vehicles. In: Proceedings of the 1st International Conference on Humans Operating Unmanned Systems (HUMOUS), Brest, France (2008)

Breslow, L.A., Gartenberg, D., McCurry, J.M., Trafton, J.G.: Dynamic operator overload: A model for predicting workload during supervisory control. IEEE Trans Hum Mach Syst **44**(1), 30–40 (2014)

Bumiller, E., Shanker, T.: War evolves with drones, some tiny as bugs. New York Times, (2011)

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

Cummings, M.L., Mitchell, P.J.: Predicting controller capacity in supervisory control of multiple UAVs. IEEE Trans Syst Man Cybern **38**(2), 451–460 (2008)

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

Damas, C., Lambeau, B., Roucoux, F., van Lamsweerde, A.: Analyzing critical process models through behavior model synthesis. In: Proceedings of the 31st International Conference on Software Engineering (ICSE), pp. 241–251. Vancouver, Canada (2009)

DSB: The role of autonomy in DoD systems. Tech. rep., Defense Science Board, Office of the Under Secretary of Defense for Acquisition, Technology and Logistics, Washington, DC (2012)

Fawcett, T.: An introduction to ROC analysis. Pattern Recognit Lett **27**(8), 861–874 (2006)

Gargantini, A., Heitmeyer, C.: Using model checking to generate tests from requirements specifications. In: Proceedings of the 7th European Software Engineering Conference and 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 146–162. Toulouse, France (1999)

Gartenberg, D., Breslow, L., Park, J., McCurry, J., Trafton, J.: Adaptive automation and cue invocation: The effect of cue timing on operator error. In: Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI), pp. 3121–3130. France, Paris (2013)

Giannakopoulou, D., Magee, J.: Fluent model checking for event-based systems. ACM SIGSOFT Softw Eng Notes **28**, 257–266 (2003)

Journal: **10515** Article No.: **0157** TYPESET DISK LE CP Disp.:**2014/11/8** Pages: **39** Layout: **Small-X**

Gray, W.D., Boehm-Davis, D.A.: Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. J Exp Psychol **6**(4), 322 (2000)
Hanke, M., Halchenko, Y.O., Sederberg, P.B., Olivetti, E., Fründ, I., Rieger, J.W., Herrmann, C.S., Haxby, J.V., Hanson, S.J., Pollmann, S.: PyMVPA: a Python toolbox for multivariate pattern analysis of fMRI data. Neuroinformatics **7**(1), 37–53 (2009)
Heitmeyer, C., Jeffords, R.: Applying a formal requirements method to three NASA systems: Lessons learned. In: Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, p 84 (2007)
Heitmeyer, C., Kirby, J., Labaw, B., Archer, M., Bharadwaj, R.: Using abstraction and model checking to detect safety violations in requirements specifications. IEEE Trans Softw Eng **24**(11), 927–948 (1998)
Heitmeyer, C., Archer, M., Bharadwaj, R., Jeffords, R.: Tools for constructing requirements specifications: The SCR toolset at the age of ten. Comput Syst Sci Eng **20**(1), 19–35 (2005)
Heitmeyer, C., Pickett, M., Breslow, L., Aha, D., Trafton, J.G., Leonard, E.: High assurance human-centric decision systems. In: Proc of the 2nd International NSF-Sponsored Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE) (2013a)
Heitmeyer, C.L., Jeffords, R.D., Labaw, B.G.: Automated consistency checking of requirements specifications. ACM Trans Softw Eng Methodol **5**(3), 231–261 (1996)
Heitmeyer, C.L., Archer, M.M., Leonard, E.I., McLean, J.D.: Applying formal methods to a certifiably secure software system. IEEE Trans Softw Eng **34**(1), 82–98 (2008)
Heitmeyer, C.L., Shukla, S., Archer, M.M., Leonard, E.I.: On model-based software development. In: Munch, J., Schmid, K. (eds) Perspectives on the Future of Software Engineering, Springer, Berlin, Germany, pp 49–60 (2013b)
Heninger, K.L.: Specifying software requirements for complex systems: New techniques and their application. IEEE Trans Softw Eng **6**(1), 2–13 (1980)
ITU: Message Sequence Charts. Recommendation Z.120, Intern. Telecomm. Union, Telecomm. Standardization Sector (1999)
Jeffords, R., Heitmeyer, C.: Automatic generation of state invariants from requirements specifications. In: Proceedings of the 6th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), pp. 56–69. Lake Buena Vista, Florida (1998)
Jeffords, R.D., Heitmeyer, C.L.: A strategy for efficiently verifying requirements. ACM SIGSOFT Softw Eng Notes **28**, 28–37 (2003)
Just, M.A., Carpenter, P.A.: Eye fixations and cognitive processes. Cogn Psychol **8**(4), 441–480 (1976)
Kira, K., Rendell, L.A.: A practical approach to feature selection. In: Proceedings of the 9th International Workshop on Machine Learning (ML), pp. 249–256. Aberdeen, Scotland (1992)
Leonard, E.I., Heitmeyer, C.L.: Program synthesis from formal requirements specifications using APTS. High Order Symb Comput **16**(1–2), 63–92 (2003)
Leonard, E.I., Archer, M., Heitmeyer, C.L., Jeffords, R.D.: Direct generation of invariants for reactive models. In: Proc. 10th ACM/IEEE Conference on Formal Methods and Models for Co-Design (MEMOCODE), pp 119–130 (2012)
Pickett, M., Aha, D.W., Trafton, J.G.: Acquiring user models to test automated assistants. In: Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference (FLAIRS), pp 112–117 (2013)
Ratwani, R., Trafton, J.G.: A real-time eye tracking system for predicting postcompletion errors. Hum Comput Interact **26**(3), 205–245 (2011)
Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychol Bull **124**(3), 372 (1998)
Rayner, K., Morris, R.K.: Do eye movements reflect higher order processes in reading? In: From Eye to Mind. Information Acquisition in Perception, Search, and Reading, North-Holland, pp 191–204 (1990)
Rothamel, T., Heitmeyer, C., Leonard, E., Liu, A.: Generating optimized code from SCR specifications. In: Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2006) (2006)
Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In: Sleeman, D.H., Edwards, P. (eds.) Proceedings of the 9th International Workshop on Machine Learning (ML), pp. 385–393. Morgan Kaufmann, Aberdeen, Scotland (1992)
Selic, B.: The pragmatics of model-driven development. IEEE Softw **20**(5), 19–25 (2003)
Sengupta, S.: U.S. border agency allows others to use its drones. New York Times, (2013)

Journal: **10515** Article No.: **0157** TYPESET DISK LE CP Disp.:**2014/11/8** Pages: **39** Layout: **Small-X**

Šuc, D., Bratko, I., Sammut, C.: Learning to fly simple and robust. In: Proceedings of the 15th European Conference on Machine Learning (ECML), pp. 407–418. Pisa, Italy (2004)

Swets, J.A.: Signal detection theory and ROC analysis in psychology and diagnostics: Collected Papers. Lawrence Erlbaum Associates, Mahawa (1996)

Uchitel, S., Kramer, J., Magee, J.: Synthesis of behavioral models from scenarios. IEEE Trans Softw Eng **29**(2), 99–115 (2003)

Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behaviour model synthesis from properties and scenarios. IEEE Trans Softw Eng **35**(3), 384–406 (2009)

US Senate: The future of drones in America: law enforcement and privacy considerations, hearing before the Committee on the Judiciary. Tech. Rep. J-113-10, Washington, DC (2013)

Whittle, J., Schumann, J.: Generating statechart designs from scenarios. In: Proceedings of the 22nd International Conference on Software Engineering (ICSE), pp. 314–323. Limerick, Ireland (2000)

Journal: **10515** Article No.: **0157** TYPESET DISK LE CP Disp.:**2014/11/8** Pages: **39** Layout: **Small-X**