Capturing Safety Requirements to Enable Effective Task Allocation between Humans and Automaton in Increasingly Autonomous Systems

Natasha A. Neogi[1]
NASA Langley Research Center, Hampton, VA, 23666

There is a current drive towards enabling the deployment of increasingly autonomous systems in the National Airspace System (NAS). However, shifting the traditional roles and responsibilities between humans and automation for safety critical tasks must be managed carefully, otherwise the current emergent safety properties of the NAS may be disrupted. In this paper, a verification activity to assess the emergent safety properties of a clearly defined, safety critical, operational scenario that possesses tasks that can be fluidly allocated between human and automated agents is conducted. Task allocation role sets were proposed for a human-automation team performing a contingency maneuver in a reduced crew context. A safety critical contingency procedure (engine out on takeoff) was modeled in the Soar cognitive architecture, then translated into the Hybrid Input Output formalism. Verification activities were then performed to determine whether or not the safety properties held over the increasingly autonomous system. The verification activities lead to the development of several key insights regarding the implicit assumptions on agent capability. It subsequently illustrated the usefulness of task annotations associated with specialized requirements (e.g., communication, timing etc.), and demonstrated the feasibility of this approach.

I. Introduction

Currently there is a shifting locus of control from humans to machines for safety critical tasks in increasingly autonomous systems. The demand for automation to perform increasingly complex and adaptive tasks in concert with humans forms a driving influence in the aviation domain. However, it is difficult to capture, even under current human-machine control paradigms, what exactly the full contribution of the human is, to the overall system safety, under nominal, off nominal and contingency conditions. Thus, identifying high level safety requirements and constraints that inform basic systems design principles in allocating taskwork to teams of humans and automation in an agnostic fashion becomes an area of vital research. These safety requirements and constraints can then be used either to assess potential systems which possess differing levels of autonomy, evaluate the efficacy of various (possibly dynamic) role allocation paradigms across a single system, or to engineer decision aiding devices, in order to enhance the collaborative human-machine decision making environment for all phases of system operation.

The issues encountered in allocating the authority and responsibility for tasks and roles that were formerly performed by humans to automation in the context of a human-machine team lies at the core of technology transition problem (i.e., transitioning increasingly autonomous systems into the national airspace). Furthermore, the ability to foster teamwork between all agents in non-traditional role allocations is necessary in order to assure system safety and efficiency[21]. In particular, contingency management for increasingly autonomous systems is regarded as a critical topic: Automation is commonly considered to perform well under nominal conditions, but poorly in undefined or unspecified situations, leading to a degradation of overall system performance, safety and trust.

In this paper, the potentially fluid nature of task, function and role assignment amongst teams of human-machine agents is examined. The ability to *translate* a task (and its associated modeling formalism) that is performed by a human into a behaviorally equivalent computational task (and modeling formalism) that is executed by a machine is illustrated. A variety of human factors task analysis techniques and computation-based task modeling formalisms are examined for their suitability for this context. They are assessed with respect to their capacity to represent

[1] Research Engineer, NASA Langley Research Center, Mail Stop 130, Hampton VA, AIAA Associate Fellow.

fundamental notions such as time, state attributes and compositionality. Qualities such as flexibility and ease of translation also influence the selection process.

The formal computational model can then be verified with respect to required properties, such as correctness of the task output or mismatches in the allocation of task authority and responsibility, often in a semi-automated fashion. This yields insight into safety-specific requirements or constraints necessary to ensure the desired emergent behavior. This work is performed in the context of enabling a safe and efficient, increasingly autonomous, cargo transportation operation that uses a commercial aircraft in a reduced crew concept[1]. Task-oriented safety requirements, or *annotations*, are then derived in this context, and their implications are discussed.

The next section of the paper provides a brief background of modeling paradigms, both for human performance and task models as well as computational task models. Section III then addresses the idea of non-traditional role allocation in increasingly autonomous systems, and outlines several possible role-set combinations, along with their implications on authority and responsibility assignments. Section IV then describes the selected modeling formalisms for the task translation task: (1) the Soar cognitive architecture[23] and (2) the Hybrid Input Output Automata (HIOA)[18] modeling formalism. A reduced crew cargo operation example is described in the following section, as well as the contingency operation of an engine out upon takeoff [24]. The transition between the nominal takeoff procedure to the contingency takeoff procedure is modeled in both Soar and HIOA, and then verified. Insights and conclusions are then drawn in the final section, and future work is described.

II. Background and Modeling Paradigms

Performance modeling is a rich and diverse field, comprised of many modeling paradigms, which may be used for specification, simulation or analysis purposes. The agent-based perspective focuses on modeling the behavior of individual elements in the system, with the resulting system behavior being an emergent property of the agents and their interactions[4]. Each agent is a discrete autonomous entity, with its own goals and behaviors, along with the capability to adapt and modify these behaviors. Agents are diverse (e.g., can be humans, computational processes, physical components, etc.) and heterogeneous decision making units whose mechanisms of interaction with other agents, along with their own behavior, must be clearly describable. Agent based modeling is primarily used for simulation purposes and has been applied in the aviation and air traffic domain[14,15]. It often employs game theoretic approaches[8,19] and is used for design space exploration[8].

A. Human Factors Task Models

Task models are logical descriptions of the activities or work to be performed in order for an agent to accomplish its goals. Nearly all task analysis techniques provide, as a minimum, a description of the observable aspects of operator behavior at various levels of detail, together with some indications of the structure of the task, and are referred to as action oriented approaches. Other techniques focus on the mental processes that underlie observable behavior, e.g., decision-making and problem solving, and are considered cognitive approaches. Hierarchical task analysis is an action-oriented approach[5], while GOMS (Goals, Operators, Methods, Selection rules)[6] is a cognitive approach. Task models and tools such as IMPRINT, may be seen as logistics models coordinating the resources and tasks, representing the individuals as nodes within the network of an organization[9,10]. There are several cognitive architectures that examine detailed cognitive mechanisms. The NASA human performance modeling project examined the ACT-R (alone and linked to IMPRINT), Air Man–Machine Integrated Design and Analysis System (Air MIDAS), D-OMAR, and A-SA formalisms in an aviation context[11]. The Brahms agent-based architecture can be used to examine multi-agent collaborative operations. In this framework, each agent is modeled around a belief–desire–intention structure defining its own practices. The models of the performed activities are embedded within the agents, and detailed negotiation protocols must be defined to encapsulate agent interaction[12,13].

The work-models-that-compute (WMTC) simulation framework represents work as a combination of resources and actions, and provides a structure to represent how work contributes to overall system goals[16]. The work model does not distinguish between task work and teamwork[7], and changes in context affect the selection of appropriate strategies and decision actions. This formalism allows for the modeling of work at multiple levels of abstraction, such as mission goals, priorities and values, generalized functions and temporal functions. This enables the evaluation of emergent behavior at a system level. This is an extremely flexible and promising architecture, but it is non-trivial to translate the work models into formal computational models, due to the informal nature of the WMTC semantics. Many similar established modeling frameworks focus on teams and teamwork, yielding a plethora of tools and methods for modeling and/or simulating roles, the distribution of authority and responsibility within a team, and communication channels[25]. These methods also lack formal semantics, and often do not adequately address the notion of continuous time and the dynamic evolution of the system.

The Soar cognitive architecture[26] allows for the representation of knowledge as formal production rules, which are expressed in first order logic. This allows for an ease of translation to computation models. Beyond most rule-based systems, Soar enables parallel associative memory, as well as belief maintenance, preference based deliberation, automatic subgoaling and decomposition via problem spaces. This flexibility, ease of translation, and compositionality was the reason the Soar architecture is used in this work. Soar also allows for adaptation via generalization of experience, or learning, which may prove valuable for future extensions of this work.

B. Models of Computation

Models of computation are as diverse and complex as human task and performance models. Computational models include descriptions of their allowable primitive operations, as well as the associated unit cost with each of these operations. Computation models include formalisms such as logic circuits, finite state machines (both deterministic and non-deterministic), Markov Decision Processes (MDPs), Turing Machines, lambda calculus and rewriting systems[2].

Due to their ability to model the dynamic transfer of control amongst human-machine teams, as well as incorporating the capacity to encompass uncertainty (e.g., human, environmental etc.) Partially Observable Markov Decision Process (POMDP) based methods for modeling increasingly autonomous systems have received much attention[27, 28]. However, since MDPs do not traditionally admit the notion of continuous time, these works have used very fine-grained discretization of time, resulting in large MDP state spaces, and prohibitive algorithm runtimes. To remedy that, continuous time MDP solvers[29-32] can be used to compute adjustable autonomy strategies. However, issues of scalability, in regards to the inability to handle multiple human operators and the lack of implicit modeling of the agent's changing responsiveness, rendered this formalism unsuitable.

Finite state machines have been used to great effect to model computational agents[33]. A finite-state machine (FSM) is a machine with memory. A FSM $M = (Q, \Theta, \Sigma, \Psi, \Delta, \Lambda, F)$ executes a series of steps during each of which it takes its current state q from the set Q of states and current external input σ from the input alphabet Σ, and uses the transition function $\Delta(q, \sigma) = q'$ to produce the successor state $q' \in Q$ and the output function $\Lambda(q, \sigma) \rightarrow \psi$ to produce ψ from the output alphabet Ψ. The sets Θ and F correspond to the sets of initial and final states, respectively. There is a host of state machine based modeling formalisms, as seen in References (2,34). However, they do not embrace a continuous notion of time. A hybrid automaton is a state machine that does not constrain the set of allowable states Q to be a finite set[18]. A hybrid automaton admits the continuous notion of time to allow states to evolve either as continuous trajectories (through differential equations), or via discrete actions. A hybrid input output automata HIOA allows for model composition, and thus scalability. It is also input enabled, and admits the use of semi-automated verification tools. This formalism was selected for this work for these reasons.

For the purposes of this work, the main point of interest is the emergent *safety* properties of the system. The goal of the work is to capture what, if any, safety effect results translating a task performed by a human agent to a task performed by an automated agent. The Soar cognitive architecture and hybrid automata model of computation are chosen for this purpose, due to their scalability and flexibility. A description of these frameworks, as well as an example of a contingency operation, can be found in Sections IV and V respectively. This next section addresses changing roles of humans and automation in the cockpit, and the task/function allocation problem.

III. Changing Roles in Human-Automations Teaming

The role of automation in the cockpit is changing; humans and machines will be interacting in novel and diverse ways under new concepts of operations (Conops) that take full advantage of the inherent abilities of both agents. Increasing safety and efficiency of pilot operations (especially single pilot operations) through the use of novel human-machine teaming paradigms is a subject of great interest. This section focuses on assessing overall system safety under novel task and human-machine role assignments in the context of reduced crew scenarios.

A. Terminology

For the purposes of this paper, *work* is defined as purposeful activity acting on a dynamic environment and in response to the demands of this environment[35, 21]. The *environment* is characterized as the combination of socio-physical constructs (including social, cultural and policy elements) essential to structuring the work dynamics, including relevant constraints[22]. A *task* can be defined as a unit of work to be performed that requires specific information, resources (e.g., physical tools, processing power etc.) and/or interactions with other agents. The task execution time may be state or sequence dependent (i.e., may depend on the preceding task executed and environmental conditions), and may vary depending on the agent(s) performing the task[34]. A *function* is a specific goal or activity that can be accomplished through taskwork[36]. *Taskwork* refers to the tasks that agents must perform

to achieve their goals[37]. *Teamwork* refers to the work required for agents to coordinate the allocated taskwork, as well as referring to activities such as inter-agent communication along with monitoring other agent's taskwork[22].

There has been a great deal of discussion surrounding the definition of the term autonomy. The International Civil Aviation Organization (ICAO) defines an autonomous operation as "an operation during which a remotely piloted aircraft is operating without pilot intervention in the management of the flight"[38]. Thus, in the context of this paper, the issue of whether or not all functions can be performed without human intervention is deferred by speaking of *increasingly autonomous systems*. In a looser sense, autonomy can be used to refer to an agent's ability to perform a task independently.

If an agent is assigned a task to execute, the agent has the authority to perform that task. If an agent is held accountable for the outcomes of a task in a legal sense (e.g., through organizational, regulatory or natural means), then that agent holds the responsibility for accomplishing the task. If the authority and responsibility for a task is not aligned (e.g., not performed by the same agent) monitoring requirements are necessary[22]. Similarly, requirements for means of intervention may also be needed for safety critical tasks.

B. Function and Role Allocation

A major focus for increasingly autonomous systems is the concept of dynamic task or function allocation in the aircraft. The idea is that the automation should be flexible in the amount of support that it supplies, and thus allow the pilot to modify the task assignments between agents in an online fashion as environmental conditions change.

The process of assigning the authority and responsibility for tasks or functions and their outcomes to individual agents, both human and automated, is referred to as task or function allocation. In addition, the allocation of taskwork functions then creates the need for additional teamwork functions to coordinate between agents. These design decisions are made considering the capability of each agent, as well as the overall teamwork required[22]. Roles define the broad areas of responsibility and authority in a language that is understood by current aircrews and give insight into the intended interaction between individuals within the team. Roles define how tasks or work is distributed in the cockpit, clarify expectations for team interactions, and establish the baseline condition from which shared situation awareness will be derived.

In this paper, several potential paradigms of interaction between the automation and the pilot were considered. The explicit roles considered for the automation were as follows: (1) monitor, (2) copilot, (3) novice pilot, and (4) full pilot. In the monitor role, the automation would perform no actuation tasks in the cockpit, and merely monitor procedure execution for off nominal or contingency situations. In the copilot role, the automation added to the functions of the observer role by performing secondary actuation tasks, similar to those performed by a copilot. Note that this is the role selected for the operational example of the engine out on takeoff contingency scenario detailed in Section V. The role of novice pilot involves the automation assuming partial responsibility for the primary flight control (e.g., throttle) and navigation of the aircraft. The human performs primary flight tasks such as communications with Air Traffic Control (ATC), and mission oriented navigation decisions, while closely monitoring and providing guidance. The full pilot role allows the automation the capability of executing the PF role at the same level of proficiency as a fully trained human pilot.

Conventionally defining the term pilot flying (PF) as the agent designated as being responsible for primary flight controls in the aircraft (e.g., stick and surface inputs), and pilot not flying (PNF) as the agent not responsible for primary flight controls then the role sets fall out as follows. In the first two role assignments, the automation is the pilot not flying (PNF), while the human is the pilot flying (PF). The reverse is true of the latter two role assignments. The point of these roles is to define general areas of responsibility during flight operations and intentionally reflect current crew resource management techniques deployed by human pilots. Note however, that the human pilot is always ultimately responsible for the overall safe execution of the flight. Thus, the pilot should be capable of overriding the automation to take control back from the aircraft if the automation is acting in an undesired manner.

The role sets describe roles only in the terms of responsibility for flight controls. In reality, operations are far more complex – there are a variety of types of tasks that are performed that must be addressed. These tasks can then be distributed among the role sets. However, in the case of an off nominal or contingency situation, the human could take any of the functions (or groups of functions) back from the automation. The utility of both the role sets and the functional groups is that they provide the pilot with multiple means of reassigning tasks both prior to and during flight. Ultimately, these role sets and functional groupings address the management of cockpit complexity, and are only of utility if they aid in this goal. Transferring tasks between the pilot and automation is not a trivial process, and involves changes in interactions among agents and information requirements. Further study must be done to determine what the feasible (and possibly optimal) role sets may be.

In this work, the automation is assigned the role of copilot in the engine out on takeoff contingency example. Both the pilot and copilot actions for the nominal takeoff and engine out on takeoff procedures are modeled in the Soar and HIOA frameworks, then verified in Section V. But first, the Soar and HIOA frameworks are described next in Section IV.

IV. Task Modeling: Soar and HIOA

The Soar cognitive architecture and the HIOA modeling formalism possess several common properties. They are both flexible and scalable frameworks, and have associated automated toolsets. They are mature frameworks, with a broad user community, enabling the sharing of models and development of new capabilities. Details of both these formalisms are provided below.

A. Soar Cognitive Architecture

A cognitive architecture possess: (1) memories for storing knowledge, (2) processing units that extract, select, combine, and store knowledge, and (3) languages for representing the knowledge that is stored and processed. Soar supports two forms of memory: short-term memory and long-term memory. Long-term memory (LTM) can be procedural knowledge encoded as rules, semantic knowledge encoded as declarative structures, and episodic knowledge encoded as episodes. Short-term or working memory (WM) is encoded as a symbolic graph structure so that objects can be represented with properties and relations. Symbolic short-term memory holds the agent's assessment of the current situation derived from perception and via retrieval of knowledge from its long-term memory. Action in an environment occurs through creation of motor commands in a buffer in short-term memory. The decision procedure then selects operators and detects impasses[39].

Soar has four architectural learning mechanisms: (1) Chunking, (2) Reinforcement learning, (3) Episodic learning, and (4) Semantic learning. Chunking automatically creates new rules in LTM whenever results are generated from an impasse. It speeds up performance and moves more deliberate knowledge retrieved in a substate up to a state where it can be used reactively. The new rules map the relevant pre-impasse WM elements into WM changes that prevent that impasse in similar future situations. Reinforcement learning adjusts the values of preferences for operators. Episodic learning stores a history of experiences, while semantic learning captures more abstract declarative statements[26].

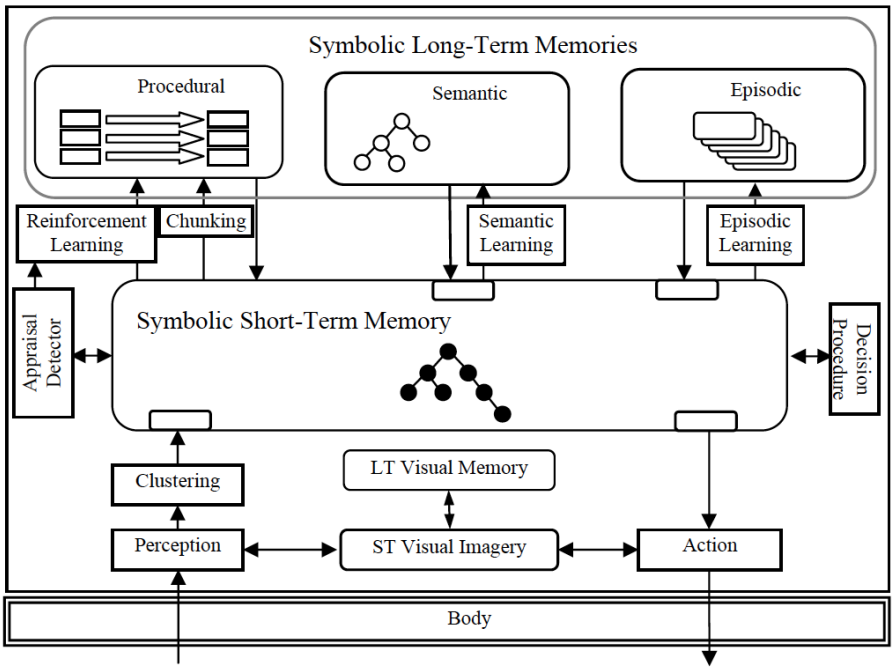

Figure 1 Soar Cognitive Architecture[26]

Basic structures supported by the Soar architecture include states (information about the current situation, goals and problem spaces), and operators (means for taking steps in problem spaces). In Soar, short-term memory is organized as graph structures in states. Each state is denoted by a unique identifier, and possesses attributes, which then have a range of values. The Perception/Motor Interface in the Soar architecture is the mechanism for defining

mappings from the external world to the internal representation in short-term memory, and from the internal representation back out to action in the external world[40].

Soar production rules, that are stored in episodic memory, are specified as follows:

$$sp\{\text{Name\_of\_Rule(state <s> ^type state)} \rightarrow \text{(write | output |) (halt))}\} \quad (1)$$

The rules are specified in the form of condition-action pairs, with each condition (on the top) or action (on the bottom) enclosed in parentheses and an implication symbol separating the group of conditions from the group of actions. Multiple conditions are permitted, as are multiple actions (as seen in Equation 1). The rule commences with "*sp*" to indicate it is a Soar production rule, and the body of the rule is always contained in parentheses. The rule is assigned an identifier (in this case, Name_of_Rule), and then the condition for the rule to fire is enumerated (in this case, that there exists a *state* variable *s* of *type* state). Vertical bars "|" indicate constants with text characters, while the symbols "<" and ">" enclose variables. A variable can match an identifier, an attribute, or a value, depending on its position in a condition – if it is first in a condition it will match the identifier, in the second position it will match the attribute, and in third it will match the value. The first condition always begins with the identifier *state*. Attributes function as links between identifiers and values, are denoted by the "^" notation, that is, the variable *s* can have an attribute (or characteristic) *type*. Note that when every Soar agent is created, it has (s1 ^type state) in working memory, which signifies that the agent does exist. Thus, the condition represented in the production rule in Equation (1) tests whether there is an identifier that possesses a value "state" (i.e., does the agent exist). If the condition is true, then the following actions are taken: the word "output" is printed in the interaction window, and then the agent is halted. The Soar cognitive architecture is used in this work because has architectural support for operators and problems spaces; higher level, abstract representations not directly supported in other rule-based systems. The result is that the Soar cognitive architecture scales much better (in both performance and the manageability of the knowledge base) than most rule-based systems, and requires fewer rules for sufficiently complex applications[39]. A key feature is that Soar supports notions of continuous time, as well as both the discrete and continuous evolution of system state and action application.

B. Hybrid Input Output Automata

A hybrid input output automata is a non-deterministic state machine that can possess an infinite number of states. An HIOA possesses state variables, which describe its system state, and may also have additional input variables and output variables. The HIOA changes state in two fashions: instantaneously through the occurrence of a discrete action, or continuously via the evolution of time. A discrete transition is defined via a triple, containing the system state, an action, and a target state. The continuous evolution of the system state is defined via system trajectories, which encapsulate the change in state variables over time, as well as input or output variables. These trajectories may be discontinuous functions[18,34].

More formally, let V be the set of variables of automaton **A**. Each $v \in V$ is associated with a (static) *type* that is the set of values v can assume. A valuation **v** for V is a function that associates each variable $v \in V$ to a value in *type* (v). The set of all valuations of V is denoted by *Val* (V). A restriction of **v** to a subset of variables $S \subset V$ is denoted by **v**.S. A trajectory τ of V is a mapping τ: $J \rightarrow val(V)$, where J is a left closed interval of time. The domain of τ is the interval J and is denoted by τ.dom. The first time of τ is the infimum of τ.dom, also written as τ.ftime. If τ.dom is right closed then τ is closed and its limit time is the supremum of τ.dom, also written as τ.ltime. Each variable $v \in V$ is also associated with a *dynamic type* (or *dtype*) which is the set of trajectories that v may follow. Dynamic type *dtype* (v) of a discrete variable v is the pasting closure of continuous constant functions from left closed intervals of time to type(v).

Thus, a hybrid I/O automaton **A** consists of:

1. A set V of variables, partitioned into *internal* X, *input* U, and *output variables* Y. The internal variables are also called state variables. The set $W = U \cup Y$ is the set of external variables.
2. A set A of actions, partitioned into internal H, input I, and output actions O.
3. A set of states $Q \subseteq val(X)$.
4. A non-empty set of start states $\Theta \subseteq Q$.
5. A set of discrete transitions $D \subseteq Q \times A \times Q$. A transition (x, a, x') $\in D$ is written in short as x$\rightarrow$x'.

6. A set of trajectories T for V, such that for every trajectory τ in T and $t \in \tau.dom$, $\tau(t).X \in Q$ and T is closed under prefix, suffix, and concatenation. The first state $\tau(0).X$ of the trajectory is denoted by $\tau.fstate$. If $\tau.dom$ is finite then $\tau.lstate = \tau(\tau.ltime).X$.

Further, the HIOA **A** is: (1) input action enabled, that is, it cannot block input actions, and (2) input trajectory enabled, that is, it accepts any trajectory of the input variables either by allowing time to progress for the entire length of the trajectory or by reacting with some internal action prior to that point. An invariant property S of the HIOA A is a condition on V that remains true in all reachable states of A. The structure of HIOA allows systematic proof of invariants. An invariant S is either derived from other invariants or proved by induction on the length of a closed execution of A as follows:

1. Base step: S(s) is true for all $s \in \Theta$,
2. Induction step: (a) discrete part: for every discrete transition $s \rightarrow s'$, I(s) implies I(s'), and (b) continuous part: for any closed trajectory $\tau \in T$, with $\tau.fstate = s$ and $\tau.lstate = s'$, I(s) implies I(s').

This structure is particularly helpful in organizing large, complex proofs and for automating invariant proofs in a theorem prover. The goal of this work is to create task models of procedures in the Soar cognitive architecture that is agnostic to which agent, either human or automation, is executing the task. Then, a translation from the readable and reviewable Soar cognitive architecture model is undertaken into the HIOA modeling formalism, whereby invariant properties of the models can be checked in a semi-automated fashion. The HIOA framework can then be used to prove properties over the composed task models that form a procedure. Properties such as agreement (between agents), validity (in task outputs), integrity (of task outputs), and task termination properties can be verified. This approach is next illustrated in Section V, in the context of a nominal takeoff procedure and an engine out on takeoff contingency procedure, for which partial Soar and HIOA models are presented.

V. Case Study: Engine Out on Takeoff Contingency Operational Scenario

For illustrative purposes, consider the scenario of a large cargo aircraft (such as a Boeing 737) during takeoff which experiences an engine failure, whereby the engine is delivering insufficient power after the aircraft brakes have been released, but before the aircraft takeoff has been successfully completed. Prior to takeoff, the speed V1 is calculated, which is defined by the FAA as "the maximum speed in the takeoff at which the pilot must take the first action (e.g., apply brakes, reduce thrust, deploy speed brakes) to stop the airplane within the accelerate-stop distance"[41]. Thus, V1 is a critical engine failure recognition speed, and can be used to determine whether or not the takeoff will continue, or result in a rejected takeoff (RTO). V1 is dependent of factors such as aircraft weight, runway length, wing flap setting, engine thrust used and runway surface contamination. If the takeoff is aborted after the aircraft has reached V1, this will likely result in a runway overrun, that is, the aircraft will stop at a point in excess of the runway. Thus, V1 is also seen as the speed beyond which the takeoff should continue: the engine failure is then handled as an airborne emergency.

Prior to the takeoff procedure, a minimum of one person qualified to operate aircraft engines must be seated in a pilot's seat when an aircraft engine is started, or running. Prior to taking the active runway for takeoff, the PF performs the following actions, and briefs the PNF with respect to: (1) special factors influencing this takeoff (wet runway, anti-icing requirements, crosswind, deviations from the norm, etc.), (2) verification of airspeed settings (bugs) and power settings, (3) verification of navigation equipment setup, (4) verification of initial flight clearance (headings, altitudes, etc.), and (5) review of the emergency return plan. Thus, there must be a shared situation awareness regarding the contingency plan at this point.

In this standard briefing of the emergency return plan, the issue of engine failure is discussed. For takeoffs that experience any warning light or reason before 80 Knots-Indicated-Airspeed (KIAS), the takeoff is aborted[24]. After exceeding this lower threshold, but before attaining V1, the takeoff is only aborted in the case of engine fire or failure, thrust reverser deployment, aircraft control problems and warning conditions.

A conventional takeoff, whereby two humans fill the roles of the pilot flying and pilot not flying proceeds as follows. Both pilots review any changes in the ATC clearance prior to initiating the Before Takeoff (BT) checklist. All Before Takeoff checklist items must be completed before the takeoff roll commences. Once the checklist is completed, the following tasks are performed (see Table 1 below). Note that the aircraft parking brake must be released on the active runway, and that the takeoff power (approximately 95% N1, which is the revolutions per minute of the low pressure spool of the engine) must be set prior to attaining 60 KIAS.

Task/Initiation Cue	PF	PNF
Takeoff		

Aircraft in position on the active runway. BT checklist completed and cleared for takeoff.	Hold brakes. Advance power to takeoff N1/Engine Pressure Rating, per AFM.	Call, "Power set, instruments stabilized." Monitor engines and systems indications.
PNF calls, "Power set, instruments stabilized."	Release brakes.	
Takeoff Roll		
Below 80 KIAS	Maintain directional control	Steady the control yoke with the right hand. (as applicable to aircraft type)
Positive airspeed indication		Call, "Airspeed alive"
PNF calls, "Airspeed alive."	Verify airspeed.	
At 80 KIAS		Verify 80 knots indicated on both PF and PNF airspeed indicators. Call, "80 knots cross-checked."
PNF calls, "80 knots crosschecked".	Move left hand from nose steering to control yoke and call, "My yoke". (as applicable to aircraft type)	
PF calls "My yoke". (as applicable to aircraft type)		Release control yoke. (as applicable to aircraft type)
At V1		Call, "V1."
PNF calls, "V1."	Move right hand to control yoke.	
At VR		Call, "Rotate."
PNF calls, "Rotate."	Rotate aircraft to pitch attitude per AFM.	

Table 1 Nominal Takeoff Procedure[24]

It can be seen that there is a great deal of interplay between the PF and PNF, especially in terms of affirming tasks and settings through callouts. These callouts also serve to initiate the subsequent task in the procedure. Thus, any tasks that are delegated to an automated PNF, performing the copilot role, must mimic this annunciation structure, in order to preserve situation awareness in the cockpit, and foster teamwork in the human-automation crew. For example, the check at 80 knots (kts) serves to perform three functions: (1) incapacitation check, (2) define the high and low speed RTO, and (3) an airspeed crosscheck. Now, in the case of an engine failure at a speed of less than V1, but above the lower threshold speed of 80 kts, the following actions are taken.

Task/Initiation Cue	PF	PNF
Engine Out Rejected Takeoff		
Engine Out Detected		The PNF closely monitors essential instruments during the takeoff roll and immediately announce abnormalities or any adverse condition significantly affecting safety of flight. Call "Engine Fire", "Engine Failure" etc.
PNF calls, "Engine Failure" below V1 (and above 80 kts)	Call "Abandon" and take control of the aircraft.	
PF calls, "Abandon"	Close thrust levers and disengage simultaneously autothrottle.	Verify thrust levers close and autothrottle disengaged. Call out omitted action items.
Autothrottle disengaged	Verify automatic RTO braking (above 90 kts) or take maximum	Note the brakes on speed. Call "Autobrake Disarm"

	manual braking (below 90 kts) as required if deceleration is not adequate or if Autobrake Disarm light is illuminated.	- Call out omitted action items.
PNF calls, "Autobrake disarm"	Raise speed brake lever.	Call "Speedbrakes Up" or Call "Speedbrakes Not Up"
PNF calls, "Speedbrakes Up"	Apply maximum reverse thrust consistent with runway conditions and continue maximum braking until certain airplane will stop on the runway.	Verify thrust reverser Call out speed: "100 Kts, 80 kts, 60 kts…"
PNF calls, "10 Kts"	Stop aircraft on runway heading or consider turning into wind if the takeoff was rejected due to fire warning.	At alternating red and white runway lights call "900 meters" of runway remaining. At steady red lights call "300 meters" of runway remaining
Aircraft stopped	Set Parking Brake	Select Flaps 40 when parking brake is set. Inform ATC including information on airplane position and alert if necessary the fire brigade

Table 2 Contingency Procedures for Engine Out During Takeoff[24]

Note that the contingency procedure is imbedded in the nominal procedure, and thus must be called from the nominal procedure. Thus, the specification to call this contingency procedure, represented in the Soar framework, would look as follows. Note that the "//" indicates a comment in the model, and that reserved keywords are in blue.

```
sp {apply*nominal-takeoff-check
(state <s> ^x 0 ^y 0 ^z 0 ^speed 0 ^runway active ^before-takeoff-check complete  ^ATC <clear> ^operator <o>)
//check to see if aircraft in nominal position on active runway & before takeoff checklist is complete
(<ATC> ^name cleared)
// check to see if ATC has cleared aircraft for takeoff
(<o> ^name nominal-takeoff-check)
//verify the operator exists, (e.g., pilot or automation, has been assigned to the takeoff checklist)
-->
// perform actions to clear first step of checklist
(verify ^name brakes ^value hold)
// verify the brakes are in the hold position
 (verify ^name power ^value 95%)
// verify N1 is set to 95%
(write|Power Set, Instruments Stabilized|)
//annunciate power has been set, instrument readings have been verified
 (monitor-engines ^engines <e>)
(<e> ^number <n> ^health <h>
//monitor the health of each engine
 (<s> ^operator <o> + )
// propose next step on checklist
(<o> ^name nominal-takeoff-check-1)
}
```

Figure 2 Soar production rule for the first row of the nominal takeoff checklist

First, we specify the first step of the nominal takeoff checklist, which is encoded in the first row of Table 1. In the first condition of the Soar model, we can see that the aircraft state is being checked so that its position (x,y,z) is on the center of the runway, that the aircraft is stationary (speed = 0), the runway is active, the before takeoff checklist has been completed, and ATC clearance is requested. The second condition checks that the ATC clearance has been received, while the third checks that the nominal takeoff checklist procedure is the procedure being executed, and has had each task assigned to an operator. The actions taken when these conditions are verified are as follows. The, the brakes are verified as being in the *hold* position, the throttle is set to the 95% N1 value, the

message “Instrument Set, Stabilized” is annunciated, and the engine health is monitored. Note that there is no specific operator bindings indicated in the task, it is only verified that there exists an operator performing each task. Thus, task allocation could easily occur by binding a specific operator (say *automation*), to a given task, in order to reflect a specific role allocation.

The monitoring function for the engine is represented in Soar as follows.

```
sp{apply*monitor-engine
(state <e> ^health Failed ^speed <s> ^condition <c> ^operator <o>)
//check to see if aircraft has a failed engine
(<e> ^speed >=80 & <=V1)
// check to see if aircraft speed is above 80 kts and below V1
(<c> ^name nominal-takeoff-checklist)
//verify that the nominal-takeoff-checklist is currently being executed
(<o> ^name monitor-engine)
//verify the operator exists for this task, (e.g., pilot or automation, has been assigned to the monitoring task)
-->
(contingency-engine-out ^engines<e>
// call contingency checklist for engine out on takeoff below V1 but above 80kts
(write|Engine Failed| <e>)
//annunciate each engine that has failed
(halt)
/halt the nominal takeoff checklist
}
```

Figure 3 Soar production rule for engine monitoring

This is the decision function that is used detect whether an engine has failed, and acts to determine whether if the engine out on takeoff contingency procedure needs to be called. Note that a separate production rule dictates what happens if an engine has failed, but the aircraft speed is above V1. The first step of the engine out contingency procedure is modeled in Soar as follows.

```
sp{apply*contingency-engine-out
(state <s> ^engine Failed ^condition <c> ^operator <o>)
//check to see if aircraft has a failed engine
(<c> ^speed >=80 & <=V1)
// check to see if aircraft speed is above 80 kts and below V1
verify nominal-takeoff-checklist ^<o> halt
/check to see that nominal takeoff procedures have been halted for all operators
(<o> ^name contingency_engine_out)
//verify the operator exists for this contingency procedure, (e.g., pilot or automation, has been assigned to the task)
-->
(<o> ^control autopilot_off)
// Decide to abort takeoff, and operator (human or automation) takes manual control of aircraft
(write|Abandon|)
//Annunciate that takeoff will be aborted
(<s> ^operator <o> + )
// propose next step on checklist
(<o> ^name contingency-engine-out-1)
}
```

Figure 4 Soar production rule for engine out on takeoff (speed below V1)

This production rule fires when the aircraft has an engine fault detected, and is still at a speed below V1 (but above 80 kts). It results in an aborted takeoff, and requires a specific bound operator (either the human or the automation) to then take control of the aircraft. An annunciation of this abort action is also required.

Production rules are generated for each step of the checklist, in order to create the Soar model of the nominal takeoff procedure, and the contingency engine out on takeoff procedure. The Soar model can then be translated into a HIOA model, for a given set of task allocation (in this case, the automation is the PNF, in a copilot role). This translation process can occur in either a semi-automated fashion (for simple rules), or by hand for more complex rules. The translation of the monitoring task (executed by the automation) is specified in the HIOA language below.

```
hybridautomaton MonitorEngine(speed, V1 :PositiveReal, accel, decel: Real, P:IndexSet)
vocabulary Monitor types
 types Phase:  Enumeration [idle, monitoring, halted]
 end

    signature
        input     Engine_1_Fail, Engine_ 2_Fail
        output  Annunciation_1, Annunciation_2, Contingency_Engine_Out

     states
        phase:Phase:= idle;
        Nominal_Takeoff_Check: Boolean := True;
        engine_health [P]: Boolean: =  True;
        ac_speed: PositiveReal:= speed;
        ac_accel: Real := accel;
        engine_out_decel: Real:= decel;
        now: Real :=0;


    transitions
       input Engine_1_Fail
            pre Engine_Health[1]= True ∧ (ac_speed>80) ∧ (ac_speed<V1) ∧(phase = idle)
            eff  phase = Monitoring;
                 Engine_Health[1] = False;

        input Engine_2_Fail
            pre Engine_Health[2]= True ∧ (ac_speed>80) ∧ (ac_speed<V1) ∧(phase = idle)
            eff  phase = Monitoring;
                 Engine_Health[2] = False;

        output Annunciation_1, Contingency_Engine_Out
            pre Engine_Health[1] = False ∧ (speed>80) ∧ (speed<V1) ∧(phase = monitoring)
            eff  phase = Halt;
                  Nominal_Takeoff_Check = False;

        output Annunciation_2, Contingency_Engine_Out
            pre Engine_Health[2]= False ∧ (speed>80) ∧ (speed<V1) ∧(phase = monitoring)
            eff  phase = Halt;
                  Nominal_Takeoff_Check = False;

        trajectories
            trajdef  idle
                invariant phase =idle;
                stop when Engine_1_Fail ∨ Engine_2_Fail;
                evolve d(now) =1;
                        d(ac_speed) = ac_accel;
            trajdef  monitoring
                invariant phase =monitoring;
                stop when Engine_Health[1] = False ∨ Engine_Health[2] = False;
                evolve d(now) =1;
                        d(ac_speed) = (ac_accel - engine_out_decel);
            trajdef  halt
                invariant phase =halt;
                evolve d(now) =1;
```

Figure 5: HIOA for Engine Monitor for Contingency Operation

This model is a direct translation of the Soar production rule seen in Fig. 3. Inputs regarding the state of each engine are received, and annunciations are output for failed engines. The contingency for engine out on takeoff is invoked if the speed conditions are met (a separate, parallel automaton would call for a contingency maneuver the aircraft speed were above V1). Note that the automaton models the continuous aircraft dynamics through trajectories, and is able to estimate the aircraft speed in the absence of sensor data. Similar automata are created for each production rule, whether the human or automation executes them, and then all of these automata are composed.

The property to be verified over the composed automata is that at no time will the contingency function not be invoked if an engine failure has been detected, and the aircraft speed is between 80 kts and V1. The automated verification activity yielded a surprising counterexample, insomuch as Soar allows for concurrent execution of actions. That is, if the automated agent was given the authority to halt the nominal takeoff procedure, but the human agent was given the responsibility of annunciating the engine failure(s), there would exist a brief time period where there was a mismatch in the system state across operators. That is, the automation would be in the state whereby the nominal takeoff procedure was halted, and the contingency procedure was being executed, while the human would still be in the state where the nominal takeoff procedure checklist was being executed (at least until the engine failure annunciations were finished). Thus, there would exist a finite closed interval of time where the engine failure had been detected, and the contingency procedure not invoked with respect to one of the operators. Hence, the proof of the invariant generated a counterexample, which can be easily fixed by requiring a dwell time (lower bound on the execution time) for the 'monitor-engine' task.

The full power of the HIOA formalism is required in order to allow for the evolution of continuous variables, such as the aircraft speed. Specifically, when a continuous action, such as 'reverse thrust' or 'braking' is applied, the dynamics of the aircraft (and of the speed variable) evolve in a continuous fashion. The HIOA formalism enables us to express state variables as differential equations, whose trajectories evolve in a continuous fashion.

Several key insights were gained through modeling the nominal takeoff and contingency engine out on takeoff procedures. They are summarized in the following section, and future avenues of research are outlined.

VI. Insights and Conclusions

A. Insights

The first key insight concerned the development of specific safety constraints, resulting from the novel function allocation and role sets described in Section III. As stated in Reference (22), function allocation must meet several crucial requirements, of which four are excerpted: (1) Each agent must be allocated functions that it is capable of performing, (2) Each agent must be capable of performing its collective set of functions, (3) The function allocation must be realizable with reasonable teamwork, and (4) The function allocation must support the dynamics of the work. Even if the assumption is made that human and automated agents execute tasks in an identical fashion, reallocation of authority (or responsibility) for a given task will likely impact the information flow between agents, especially if a monitoring task is generated due to an authority-responsibility mismatch. More critically, if situation awareness is not fostered between agents, and an inconsistent state arises due to information latency or incorrectness, agents may execute conflicting tasks. For instance, if a contingency is detected by one agent prior to the other agent's detection, and there is insufficient communication (teamwork) or trust between agents, it is possible for agents to both believe they have authority for a given function (e.g., primary flight control) for a short period of time (and potentially during any hand-off situation).

A second insight was derived through the verification activities that occurred on the Soar and HIOA models. It became necessary to create specific safety oriented requirements and constraints for tasks and roles sets related to the functions of perception, actuation, communication and timing. These were necessary refinements of the first two requirements in the previous paragraph, which were used to guarantee the feasibility of the task assignments. These requirements were discovered throughout the verification activity, whereby counterexamples were encountered due to inconsistent system and agent states. These safety requirements were often conservative in nature, such as enforcing a minimum dwell time in a given task, in order for one agent to not get ahead of another agent in a series of interleaved tasks. While this enforces the correct execution order, it acts to reduce efficiency in the cockpit, and may impact overall teamwork if one agent is perceived as waiting on another.

Perception requirements are especially relevant to safety critical task execution. Agents perceive the world differently, be they human or automation. Additionally, different sources of information are accorded different levels of integrity depending on the nature of the agent, and thus influence decision making in an asymmetric fashion. For example, in the case of an engine fire, an automated agent will accord more weight to a sensor reading, while a human agent is more likely to trust any visual cues that are present. Moreover, while perception (and information) aid in fostering situation awareness, they are not identical constructs. Thus, a human agent and an

automated agent can have identical states and be immersed in the same environment, yet believe themselves to be in different situations. Thus, while a human pilot relies on a copilot to annunciate his previously taken action to confirm its execution, an automated system believes a task to be complete when it has implemented the action. Thus, annotating safety critical tasks with perception requirements, identifying sources of perception information, levels of information integrity, and feedback requirements, is essential.

The generation of actuation requirements is often related to the physical capabilities of the executing agent. They can be tied into issues related to timing, whereby a human agent might apply an actuation input (e.g., continuous throttle) in a different fashion than an automated agent (e.g., step input throttle). Alternatively, reaction times differ between human agents and automated agents for specific tasks; this also depends on the agent's level of expertise.

Similarly, communications requirements for safety critical tasks may be generated when interleaved task executions are permitted. Since multiple agents execute in parallel, it would be extremely inefficient to limit task execution to a serial fashion. Therefore, it is necessary for the agents performing safety critical roles, for which they may not have previously held the authority, to annunciate both completed actions, in order to verify the execution, and the intent to perform any safety critical action. Likewise, if a series of tasks are interdependent, it is efficacious to have a single agent possess the authority for these tasks, as additional communications requirements are generated for safety critical tasks, as well as potential monitoring requirements. Interdependent tasks should be grouped together by functionality, and have their authority and responsibility assigned to a single agent in a single role set, in order to limit the number of communications requirements.

Timing requirements are perhaps the most significant, as they touch on all of the previous requirements issues. Time is a factor that directly affects the feasibility of a task or function allocation, as humans and automation deal with a preponderance of deadlines occurring at the same time in a different fashion. Furthermore, due to the dynamic environment in which the taskwork takes place, it is important to be able to assess workload, worst-case execution time, and environment or state dependent influences on how long it may take a task to complete. The execution of an identical sequence of tasks will take different times depending on the expertise of the agent, as well as on the number of handoffs in authority and responsibility occurring in the sequence. A common timing requirement on safety critical tasks is that they must be given sufficient time to execute, and must terminate correctly in a bounded of time. Without this guarantee, safety critical monitoring or decision aiding tasks may worsen the situation they were designed to mitigate. Finally, many important timing attributes may only be revealed during actual operations, particularly when a function allocation requires strongly coupled interplay between agents.

B. Conclusion and Future Work

This work seeks to safely enable increasingly autonomous systems by aiding in the development of requirements that facilitate the transfer of authority and responsibility for tasks from human agents to automation in a provably safe manner. The authority and responsibility assignments associated with any task allocation paradigm should leverage the inherent ability (and model structure) of individual agents, thereby acting to increase the fault-tolerance, resiliency and safety of the overall system. This issue was examined by conducting a verification activity to assess the emergent safety properties of a clearly defined, safety critical, operational scenario that possesses tasks that can be fluidly allocated between human and automated agents. The work was done in the context of a reduced crew operational concept for cargo transportation. This required that a non-traditional task allocation (including novel authority and responsibility assignments) be developed for the procedures executed in the operational scenario, and a spectrum of roles for the human and automation were proposed. A standard pilot-copilot role set was selected for the human-automation team, and a safety critical contingency procedure was modeled, in the form of an engine out on takeoff. Both the nominal takeoff and contingency engine out on takeoff procedures were modeled in the Soar cognitive architecture and HIOA formalisms, in order to evaluate the monitoring task that alerts for the selection of the contingency procedure. Verification activities were then performed to determine whether or not the safety properties of the conventionally piloted procedures transferred to the increasingly autonomous system. The verification activities lead to the development of several key insights regarding the implicit assumptions on agent capability. It subsequently illustrated the usefulness of task annotations associated with perception, actuation, communication and timing requirements, and demonstrated the feasibility of this approach.

A few caveats must be considered in light of these insights. The assessment case study of a contingency operation used operational procedures adapted from conventionally piloted aircraft. There has been no evaluation of whether or not these baseline procedures are necessary and/or sufficient to ensure safety in an increasingly autonomous system, with reduced crew operations. While the verification activities on the HIOA and Soar models of these scenarios enabled an assessment of the transition of authority and responsibility for specifically identified

safety critical functions (such as the selection of a relevant contingency strategy etc.) between a human pilot and an automated agent, there has been no experimental study or workload analysis for the posited role set.

This analysis is not performed with the goal of asserting what the optimal functional allocation might be, but rather with the aim of capturing and specifying the relevant safety properties that should be verified for the flight deck automation to operate in a provably safe manner in its operational context. These requirements may also be used to inform the design of decision aiding devices.

The use of the Soar cognitive architecture allows for the incorporation of learning strategies. This will potentially enable a wide range of operational concepts involving increasingly autonomous systems to be assured in this framework. Soar allows for the aggregation (chunking) of production rules, in order to learn from previous situations it has experienced. Thus, these learning systems can potentially be continuously re-verified in this framework. Future extension of this work involves creating an automated process to translate Soar production rules into HIOA models. Currently, only the simplest formulae can be translated from Soar, through an intermediate theorem prover, to HIOA. Additionally, the representation achieved from this translation process is not necessarily unique, which can lead differing runtimes taken to discharge proofs of invariants. Avenues for standardizing this process, using a template approach, exhibit promise. Developing an automated process to translate the altered production rules and re-verify the HIOAs could be very advantageous in providing assurance for learning systems.

Finally, a potential research direction involves investigating a mapping between the decision spaces of the human agent and the automated agent performing the same task. Points of discontinuity between the decision spaces may yield insights into safety critical task requirements, as well as shed light on the overall human contribution to safety.

Acknowledgments

The author would like to thank Dr. Siddhartha Bhattacharyya at Rockwell Collins Inc., for his invaluable help in working with the Soar cognitive architecture. The author would also like to thank Natalia Alexandrov and Paul Schutte, from NASA Langley Research Center, for their invaluable input and advice. This work was performed under the NASA Airspace Operations and Safety Program (AOSP) Safe Autonomous Systems Operation (SASO) project.

References

[1]Vatistas, G. H., Lin, S., and Kwok, C. K., "Reverse Flow Radius in Vortex Chambers," *AIAA Journal*, Vol. 24, No. 11, 1986, pp. 1872, 1873.

[1]Bilimoria, K., Johnson, W., Schutte, P., "Conceptual Framework for Single Pilot Operations", International Conference on Human-Computer Interaction in Aerospace (HCI-Aero 2014), July 30 – August 1, 2014, Santa Clara, California, USA.

[2]Sipser, M., *Introduction to the Theory of Computation*, 1st ed., International Thomson Publishing, 1996.

[3]Savage, J. E., *Models of Computation: Exploring the Power of Computing*, 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[4]Macal, C., and North, M., "Tutorial on Agent-Based Modelling and Simulation," Journal of Simulation, Vol. 4, No. 3, 2010, pp. 151–162.doi:10.1057/jos.2010.3.

[5]Paterno, F., "Task Models in Interactive Software Systems", https://www.cefns.nau.edu/~edo/Classes/CS477_WWW/Docs/TechArticles/Task-models-in-UI.pdf

[6]John, B., Kieras, D., "The GOMS Family of Analysis Techniques: Comparison and Contrast", ACM Transactions on Computer-Human Interaction, Vol.3, N.4, pp.320-351, 1996.

[7]Helmreich, R. and Foushee, F., "Why CRM: Empirical and Theoretical Bases for Human Factors Training", Elsevier, 2002.

[8]Rouse, W., and Boff, K., (eds.), Organizational Simulation, Wiley, Indianapolis, IN, 2005.

[9]Mitchell, D., Samms, C., Henthorn, T., and Wojciechowski, J., "Trade Study: ATwo-Versus Three-Soldier Crew for the Mounted Combat System (MCS) and Other Future Combat System Platforms," U.S. Army Research Lab. TR-ARL-TR-3026, Sept. 2003.

[10]Allender, L., "Modeling Human Performance: Impacting System Design, Performance, and Cost," Simulation Series, Vol. 32, No. 3, 2000, pp. 139–144.

[11]Foyle, D. C., and Hooey, B. L., Human Performance Modeling in Aviation, CRC Press, Boca Raton, FL, 2008, pp. 3–64.

[12]Clancey,W. J., Sachs, P., Sierhuis, M., and van Hoof, R., "Brahms: Simulating Practice for Work Systems Design," International Journal of Human-Computer Studies, Vol. 49, No. 6, 1998, pp. 831–865. doi:10.1006/ijhc.1998.0229

[13]Sierhuis, M., Clancey,W. J., and van Hoof, R., "Brahms: A Multiagent Modeling Environment for Simulating Work Practice in Organizations," International Journal for Simulation and Process Modeling, Vol. 3, No. 3, 2007, pp. 134–152.

[14]Shah, A. P., Pritchett, A. R., Feigh, K. M., Kalaver, S. A., Corker, K. C., and Jadhav, A., "Analyzing Air Traffic Management Systems Using Agent-Based Modeling and Simulation," 6th USA/Europe ATM R&D Seminar, Baltimore, MD, June 2005, http://www.atmseminar.org/papers.cfm?seminar_ID=6 [retrieved 6 Oct. 2014].

[15]Pritchett, A. R., Lee, S. M., Abkin, M. H., Gilgur, A. Z., Bea, R. C., Corker, K. M., Verma, S., and Jadhav, A., "Examining Air Transportation Safety Issues Through Agent-Based Simulation Incorporating Human Performance Models," Proceedings of the 21st Digital Avionics Systems Conference, Vol. 2, IEEE, Piscataway, NJ, 2002, pp. 1–13.

[16]Pritchett, A. R., Feigh, K. M., So Young Kim, and Kanna, S. K., "Work Models that Compute to Describe Multiagent Concepts of Operation: Part 1", Journal of Aerospace Information Systems, Vol. 11, No. 10, October 2014.

[17]Hayhurst, K.J., Maddalon, J. M., Miner, P. S., Szatkowski, G. N., Ulrey, M. L., DeWalt, M. P., Spitzer, C. R., "Preliminary Considerations for Classifying Hazards of Unmanned Aircraft Systems", NASA Technical Manuscript, NASA/TM-2007-214539, February 2007, pp. 11-12.

[18]Lynch, N. A., Segala, R., and Vaandrager, F., Hybrid I/O automata. Inf. Comput. 185, 1 (August 2003), 105-157. DOI=http://dx.doi.org/10.1016/S0890-5401(03)00067-1

[19]Parsons, S. D., , Gymtrasiewicz, P., Wooldridge, M. J., "Game Theory and Decision Theory in Agent Based Systems", Springer Science and Business Media LLC., 2002.

[20]Faber, A., "Single Pilot Commercial Operations: A Study of Technical Hurdles", M.Sc Thesis, Delft University of Technology, August 2013.

[21]Pritchett, A. R., S. Young Kim, and K. M. Feigh, "Modeling Human-automation Function Allocation", Journal of Cognitive Engineering and Decision Making, 2013.

[22]Pritchett, A., Kim, S. Y., Kannan, S. K., & Feigh, K. M., "Simulating situated work", In Proceedings of the IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2011, pp. 66–73.

[23]Laird, John E., *The Soar Cognitive Architecture*, MIT Press, 2012.

[24]Boeing 737 Pilots Operating Handbook, http://www.b737.org.uk

[25]Parasuraman, R., and Riley, R., "Humans and automation: Use, misuse, disuse, abuse", In Human Factors, 39, pages 230-253, 1997.

[26]Laird, J. E., "Extending the Soar Cognitive Architecture", Artificial General Intelligence Conference, Memphis, TN, March 1-3, 2008.

[27]Varakantham, P., Maheswaran, R., and Tambe, M., "Exploiting belief bounds: Practical pomdps for personal assistant agents", In AAMAS, 2005.

[28]Poupart, P., Exploiting structure to efficiently solve large-scale partially observable Markov decision processes. Diss. University of Toronto, 2005.

[29]Boyan, J., and Littman, M., "Exact solutions to time-dependent MDPs", In NIPS, pp. 1026-1032, 2000.

[30]Li, L., and Littman, M., "Lazy approximation for solving continuous finite-horizon MDPs", In AAAI, pp. 1175-1180, 2005.

[31]Marecki, J., Koenig, S., and Tambe, M., "A fast analytical algorithm for solving markov decision processes with real-valued resources", In IJCAI, January 2007.

[32]Schurr, N., Marecki, J., and Tambe, M., "Improving adjustable autonomy for time-critical domains", In Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems, Hungary, 2009. ACM.

[33]Wagner, F. and Schmuki, R. and Wagner, T. and Wolstenholme, P., Modeling Software with Finite State Machines: A Practical Approach, CRC Press, 2006.

[34]Naseri, A., and N. Neogi, "Stochastic Hybrid Models with Applications to Air Traffic Management", AIAA Guidance, Navigation, and Control Conference, Keystone, CO, August 21-24, 2006.

[35]Vicente, K. J. *Cognitive work analysis: Towards safe, productive and healthy computer based work*. Mahwah, NJ: Lawrence Erlbaum., 1999.

[36]Laughman, D., Functional Requirements Analysis and Functional Allocation, Technical Report MPWR-TECR-005004, (http://www.nrc.gov/docs/ML1216/ ML12165A704.pdf)

[37]Bolton, M. L., Siminiceanu, R., & Bass, E. J., "A systematic approach to model checking human-automation interaction using task analytic models", IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2011, pp. 961–976.

[38]International Civil Aviation Organization, Unmaned Aircraft Systems (UAS), Cir 328, 2011.

[39]Soar Tutorial 9.4.0, http://soar.eecs.umich.edu/articles/downloads/soar-suite/105-soar-tutorial-9-4-0

[40]Lehman, J. F., Laird, J. E., and Rosenbloom, P., "A gentle introduction to Soar, an architecture for human cognition", In S. Sternberg & D. Scarborough (Eds), Invitation to Cognitive Science, MIT Press, 1996.

[41]Code of Federal Regulations – Chapter 14.1, Federal Aviation Administration. http://www.ecfr.gov/cgi-bin/text idx?c=ecfr&SID= f553dbb9c 06409a040f3d6865e435c70&rgn=div8&view=text&node=14:1.0.1.1.1.0.1.2&idno=14;cc=ecfr