Toward a Multi-method Approach to Formalizing Human-automation Interaction and Human-human Communications

Dennis Griffith*, William Mansky*, Ellen J. Bass†, Matthew L. Bolton‡,
Karen Feigh§, Elsa Gunter*, and John Rushby¶

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, {dgriffi3,mansky1,egunter}@illinois.edu
†Department of Systems and Information Engineering, University of Virginia, Charlottesville, VA 22904, ejb4n@virginia.edu
‡San José State University Research Foundation, NASA Ames Research Center, Moffett Field, CA 94035, matthew.l.bolton@nasa.gov
§School of Aerospace Engineering, Georgia Institute of Technologie, Atlanta, GA 30313, kfeigh@isye.gatech.edu
¶Computer Science Laboratory, SRI International, Menlo Park, CA 94025, rushby@csl.sri.com

***Abstract*—Systems including human and automated agents require methods for verifying and validating that the roles and responsibilities potentially assignable to the human and automated agents do not lead to unsafe situations. Such analyses must consider the conditions that could impact system safety including human behavior and operational procedures, methods of collaboration and regulations. The use of task behavior as part of a larger, formal system model is potentially useful for analyzing such problems because it allows the ramifications of different human behaviors to be verified in relation to other aspects of the system. A component of task behavior largely overlooked to date is the role of human-human interaction, particularly human-human communication in complex human-computer systems. We are developing a multi-method approach based on extending the Enhanced Operator Function Model language to address human agent communications (EOFMC). This approach includes analyses via theorem proving and model checking linked through the EOFMC top level XML description. Herein, we consider a continuous descent arrival (CDA) scenario including the roles of the pilots monitoring and flying. We then show how we can use the semantics of this extended language to verify properties involving collaborations among these agents, and associated computer systems, necessary to guarantee safety in the CDA context.**

***Index Terms*—Keywords goes here.**

I. Introduction

Failures in complex, safety-critical systems can arise as a result of interactions between the elements of the system, including its human operators. The human communication processes, including human-human communication and human-automation interaction, are important to the operation of safety critical systems but have contributed to failures in complex systems in a number of domains, including aviation [1], [2], [3], [4]. The use of task behavior as part of a larger, formal system model is potentially useful for analyzing such safety-critical systems because it allows the ramifications of different human behaviors to be verified in relation to other aspects of the system. The Enhanced Operator Function Model (EOFM) [5], [6] is one method of specifying human behavior in a way amenable to formal analysis. We are developing a multi-method approach where analyses via theorem proving and model checking are linked through a top-level XML description of human task behavior. In this paper, we consider a continuous descent arrival (CDA) scenario and propose an extension to the EOFM language to address human-human communications, and show how we can use the semantics of this extended language to verify properties of these agents necessary to guarantee safety. We have chosen to focus upon the pilots' roles in CDA, and particularly upon the pilots' roles in assuring that corrective action is taken if the speed of the aircraft begins to exceed the expected speed.

II. Continuous Descent Arrival

Continuous Descent Arrival (CDA) is a procedure in which aircraft descend from cruise altitude directly onto the Instrument Landing System (ILS) glide slope without leveling out at intermediate altitudes during the descent (see the black line in Figure 1). By eliminating the level altitude segments and their associated thrust transients at low altitude, aircraft are kept higher and at lower thrust prior to intercepting the ILS, thereby improving fuel economy and reducing noise.

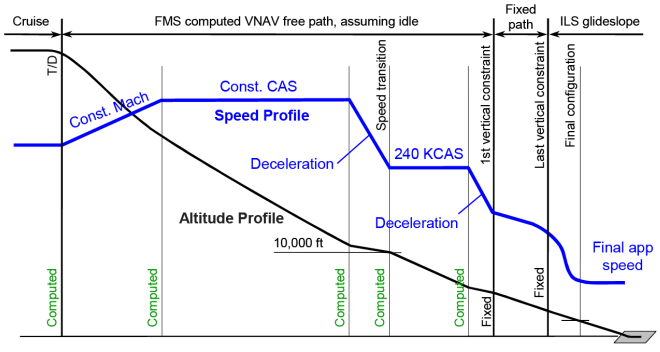

Fig. 1. Vertical Profile of RNAV CDA ©(Ren & Clarke, 2007)

RNAV CDA is an advanced type of CDA that involves a pre-defined trajectory of a series of waypoints with altitude and/or speed targets as required. These waypoints and constraints

can be programmed into an aircraft's area navigation (RNAV) equipment such as the Flight Management System (FMS). A CDA procedure starts at cruise altitude and continues along the FMS-computed VNAV descent path until the Final Approach Fix. Thus, when the CDA procedure ends, the aircraft is established on the ILS localizer and glide slope. The FMS LNAV (lateral navigation) function computes the lateral path from the waypoints and, with the autopilot, maintains the aircraft on the computed lateral path. The FMS VNAV (vertical navigation) function computes the vertical and speed profiles to satisfy associated constraints if possible, and commands the autopilot and autothrottle to follow the computed vertical and speed profiles.

A. Role of the Pilots

In addition to communication, the main objectives of the pilots during CDA are to ensure that the aircraft follows the specified path and speed constraints while maintaining safety, legality, and passenger comfort. They must ensure operationally feasible and stable flight control while executing descent and before landing activities. Because aircraft are continuously descending, variations in aircraft trajectories caused by differences in aircraft performance make it difficult for air traffic controllers to predict future spacing between aircraft and manage the arrival flow. In order to assure adequate separation, ATC provides CDA procedure clearances and the pilots must maintain horizontal and vertical path and speed errors to within preset bounds. The pilots must ensure sufficient margins to cope with vertical and horizontal path uncertainty caused by environmental features such as the wind. The assurance of adequate safety margins can be achieved by selecting CDA parameters such as the altitude and velocity at the top of descent point and the altitude at which the thrust will be set to idle. These parameters are selected to allow aircraft to operate within their operational envelope and have sufficient time to decelerate to the target altitude and speed.

While considering the aircraft's state, the pilot flying is responsible for the activation of the non-automated equipment such as flaps, speed brakes, and landing gear subject to equipment integrity concerns and based on flightpath maintenance and checklist procedures. The use of speed brakes, for example, may be necessary when additional drag is needed to keep the aircraft following a given path profile such as when the tailwind experienced by the aircraft is stronger than the wind used by the FMS when the VNAV path is computed. It should be noted that, while the speed brake can be used to manage the deceleration of the aircraft by increasing the drag, such utility undermines its usage as a compensatory device for contingency situations. Other reasons that make the usage of speed brakes undesirable include additional airframe noise and vibration in the cabin, which would be uncomfortable for the passengers.

B. Role of the Controller

The air traffic controller's role divides into four phases (see Figure 2):

1) Streaming and sequencing
2) Spacing
3) Monitoring and intervention if separation is projected to be violated and
4) Missed approach

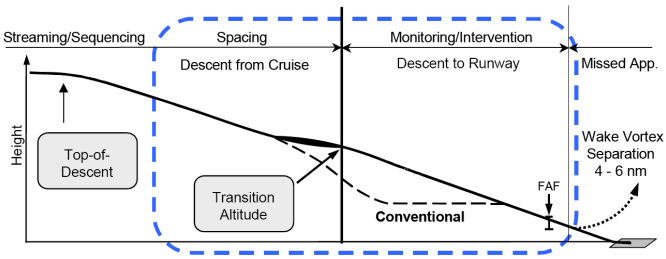

Fig. 2. Proposed CDA Procedure Design and Operational Framework ©(Ren, 2007)

A conceptual intermediate metering point separates the descent from cruise and the low noise descent to the runway. A target spacing between consecutive aircraft is specified for the intermediate metering point such that separation can be assured (subject to uncertainty) without further controller intervention. Before the metering point, the controller is free to vector aircraft as necessary to ensure separation and appropriate initial speeds at the start of the CDA procedure. With these initial conditions properly set, aircraft can continue the CDA without further vectoring from the controller until established on final approach. Additional spacing can be achieved by 1) adjusting speed, 2) vectoring the aircraft off the procedure lateral path and returning it to the path when the required separation is reestablished, 3) extending the base leg, or 4) by sidestepping to an alternate runway if it is available. As the aircraft approaches the Final Approach Fix (FAF), the flight will be cleared in a conventional fashion. Then, the controller only intervenes when missed approach is necessary.

In the RNAV CDA, tactical control is removed from the controller because detailed trajectory intent information available to pilots through the FMS is not available to the controller. Controllers are only given discrete control over the aircraft trajectories performing CDA, such as the authorization to begin a procedure (e.g., "cleared ILS 4L"). The controllers clear the aircraft to begin the approach, and the pilot then executes the FMS controlled approach. If controllers determine that action should be taken to prevent a conflict, the controller can remove the aircraft from the CDA approach and resume a conventional approach or command the aircraft to perform a go-around procedure.

C. Role of the Flight Management System Automation

The Flight Management System (FMS) has two major functions. Before CDA execution, the FMS builds the lateral flight path (LNAV path) based on the horizontal location of the given waypoints, and the vertical flight path (VNAV path) based on the given altitude/speed constraints at specified waypoints. During the execution of the procedure, the FMS continuously monitors the state of the aircraft and compares it with the

computed LNAV path and VNAV path. It selects the autopilot and auto-throttle control modes by considering the calculated flight path, altitude and speed constraints, and altitude, speed, and cross-track error. The FMS requires a continuous lateral path from cruise to the point where the aircraft is established on the final approach glide slope.

D. Safety Properties

A major concern during continuous descent arrival is that the aircraft have a trajectory that is predictably within specified bounds. This is necessary for being able to guarantee appropriate separation of aircraft. It is also necessary to ensure that, as the aircraft approaches the Final Approach Fix, it will be on altitude, speed, and horizontal position so that it can be cleared for landing without extra vectoring or speed/altitude adjustments except those associated with landing. As mentioned above, the primary tools available to the pilots to adjust the vertical profile are the flaps and speed brakes. Using a formal model of the CDA procedure, we have begun to analyze the process by which the pilots notice the need for course correction and make use of these tools (particularly the speed brakes) to maintain the desired trajectory.

III. EOFM with Communication (EOFMC)

A. Overview and Syntax

In order to model the task behavior of the human operators in CDA, we make use of a variant of EOFM which we call EOFMC. EOFM is an XML-based, platform- and analysis-independent language for describing task analytic models [6]. The EOFM language allows for the modeling of a human operator as an input/output system. Inputs may come from the human-device interface, environment, mission goals, and other human operators. Human actions are outputs. The operator's task model describes how human actions are generated based on input (from device interfaces, the environment, and human communications) and local variables (representing perceptual or cognitive processing). When translated for use in model checking, instantiated EOFM models can be integrated into formal system models. Input, local and output variables updated in different steps reflect the state of the human operator, automation, or other elements of the system.

Each human operator model is a set of EOFM task models that describe goal-level activities. Activities decompose into lower level activities and eventually atomic human actions. Activity decompositions are controlled by decomposition operators that specify the cardinality of and temporal relationship between the subactivities or actions:

- or_seq and or_par – one or more sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- optor_seq and optor_par – zero or more sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- and_par and and_seq – all sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- xor exactly one sub-activity or action must execute; and
- ord all sub-activities or actions must execute one at a time in a specific order.

EOFM activities can have preconditions, repeat conditions, and completion conditions (Boolean expressions written in terms of input, output, and local variables and constants) that specify what must be true before an activity can execute (precondition), when it can execute again (repeat condition), and what is true when it has completed execution (completion condition). EOFM atomic actions are either an assignment to an output variable (indicating an action has been performed) or a local variable (representing a perceptual or cognitive action). All variables are defined in terms of constants, user-defined types, and basic types.

EOFM's notation allows for the definition of global constants (*constant* nodes) and user-defined types (*userdefinedtype* nodes). It then allows multiple human operators to be defined (*humanoperator* nodes). Each human operator defines variables representing inputs from the human-device interfaces and environment (*inputvariable* nodes), local variables (*localvariable* nodes), and human action outputs (*humanaction* nodes). Then, a *humanoperator* node contains one or more *eofm* nodes, each defining a goal-directed task: a hierarchy of *activity* nodes and *action* nodes which define human task behavior. *activity* nodes contain conditions and decomposition operators which control how a task can execute. In the original implementation, *action* nodes occur at the bottom of the hierarchy and reference *humanaction* nodes (indicating that a human action is performed) or allow values to be assigned to local variables.

Our analysis of CDA in this work is based on EOFMC, an extension of EOFM updated to support communication actions as well as actions that set specific values via a human-device interface. Changes to the notation occurred in two contexts (Figure 3). In the first (Figure 3(a)), the notation for defining human action outputs (*humanaction* nodes) was updated to allow for an additional type of behavior called *setvalue*. Originally, human actions were treated as binary (either being performed (*true*) or not being performed (*false*)). They could exhibit two behaviors: *autoreset*, where the action would be performed (become *true*) and then automatically transition to *false*; and *toggle*, where the performance of the action would toggle it between *true* and *false*. When a human action has the new *setvalue* behavior, the modeled operator can set a specific value in a single action. This type of human action has an additional attribute specifying the type of the value being set (either a reference to a *userdefinedtype* or a *basictype*) or a reference to a local variable which will contain the value to be set.

The second change impacts how actions are specified (*action* nodes; Figure 3(b)). In its original form, only two types of actions could be performed: a *humanaction* (a reference to a *humanaction* node) or an assignment to a local variable

(*localvariable*). The new modifications provide additional action possibilities. An optional data field was added so that an action referencing a *humanaction* could set a value (*setvalue* behavior). Further, there is now a *communication* action, which allows the human operator to communicate the value contained in a referenced local variable, and a *receivewith* action that supports receiving a communication with a referenced local variable.

We have developed an XML-based syntax for EOFMC that is an extension of the syntax of EOFM. We have developed a RelaxNG specification that accepts expressions conforming to this grammar, and checks that all variables are declared before use. The formal semantics of EOFM were given by an automated translation into the SAL model checker [6], [16]. As discussed in the following section, EOFMC has been given formal transition semantics in the theorem prover Isabelle. To support formal reasoning about EOFMC in Isabelle, we have given a more compact abstract syntax for EOFMC (summarized in Figure 4). There is a precise correspondence between the abstract syntax of EOFMC and the RelaxNG specification. From an EOFMC task behavior in the abstract syntax, we have a procedure that automatically generates an XML document that conforms to the RelaxNG specification.

B. Semantics

Since EOFMC activities are specified as hierarchical trees, we use tree structures to model the semantics of tasks. In a task tree, internal nodes correspond to the specifications of activities, and leaves correspond to atomic actions. During execution of a task, each node in the tree is in one of three states: Ready, Executing, and Done. A Ready task may begin executing as soon as all necessary conditions are met; an Executing task is in progress or waiting to be allowed to complete; a Done task has completed, and may return to the Ready state if its specification allows it to reset. Each node moves between these three states in a manner governed by both its explicit conditions (precondition, completion condition, repeat condition) and a set of implicit conditions, called the *start*, *end*, and *reset* conditions, that are determined by the decomposition operator of the task and the states of parent, sibling, and child tasks. For instance, if a task is part of an ordered (ord) decomposition, its start condition will ensure that it cannot begin to execute until the previous task in the order has completed. The details of the implicit conditions and their effects on state transitions are the same as in the previous presentation of EOFM [6], with one slight revision: an activity cannot transition from Ready to Done unless its start and end conditions are both satisfied.

A complete EOFMC specification (represented by the *eofms* non-terminal in Figure 4) is modeled by a global environment env, containing the values of any defined constants, and a set of human operators $[h_1, \ldots, h_m]$. Each human operator has a name n, a local environment $lenv$ that stores the values of its local variables, and a collection of task trees $[t_1, \ldots, t_l]$, one for each top-level activity in the operator's specification. The initial task tree for an activity is generated by converting each of its sub-activities to the task tree representation, and setting each one to Ready. The formal semantics of this task tree are then given by its evolution over time: in each time-step one or more activities may advance, and one or more actions may occur. Activities advance in accordance with their state and constraints, moving from Ready through Executing to Done as allowed by their conditions.

The real work of a task is done by its actions, the leaves of the tree, which, as shown in Figure 4, come in four basic types. Figure 5 gives some of the transition rules associated with actions. As shown by the first rule, a human action humanaction(a, v) causes the operator to perform operation a with value v on the environment, provided its context $t[]$ (a task tree with a hole in it for the action) satisfies the *start* condition at the hole; the effects of this action are left to the specification of the environment. In the second rule, a local variable action localvariable(x, v) causes the operator to associate the value v with the variable x in his/her local environment, modeling the action of remembering or making a note of a value. The behavior of the other two actions, communication and receivewith, is given by the third rule. They are executed in matched pairs: when an EOFMC specification contains a human operator c with a task tree t that is ready to perform communication(v), and a collection of operators $h_{i_j},\ j = 1 \ldots, k$ with task trees t_{i_j} that are each ready to perform receivewith(x_{i_j}), then c and all the h_{i_j}'s execute, and each h_{i_j} stores the value v in his/her local variable x_{i_j}. The last rule tells us that any action that is Executing may immediately transition to being Done; an action's *end* condition is always true. Using these rules, and more for activities and *eofms*, we can give EOFMC specifications a labeled transition semantics, where the label on a transition indicates the set of human actions performed in that step.

An *execution* of a specification is defined as a series of labeled transitions beginning with the specification's initial task tree and consistent with the transition rules. We can state and verify various properties on such an execution, for instance, that action B is never performed before action A. We say that a property holds for a specification if it holds for all possible executions of that specification. Thus, given an EOFMC specification of a task behavior, we can use this model to prove safety properties of the task behavior. In addition, there is a precise relationship between our formal semantics (expressed as transition rules) and the translation to SAL given for earlier versions of EOFM [6]. For specifications not involving human-to-human communication, the transition semantics can be shown to give rise to the same sequences of human actions as the SAL translation. In this sense, the semantics for EOFMC are an extension of those for EOFM.

IV. CDA in EOFMC

A. Process

Our case study developed according to a general process for constructing formal models (specifically in EOFMC) of human-machine interaction. First, we determine the human operators to be modeled. In the case of CDA, as shown in the next section, we chose to model the behavior of the pilot

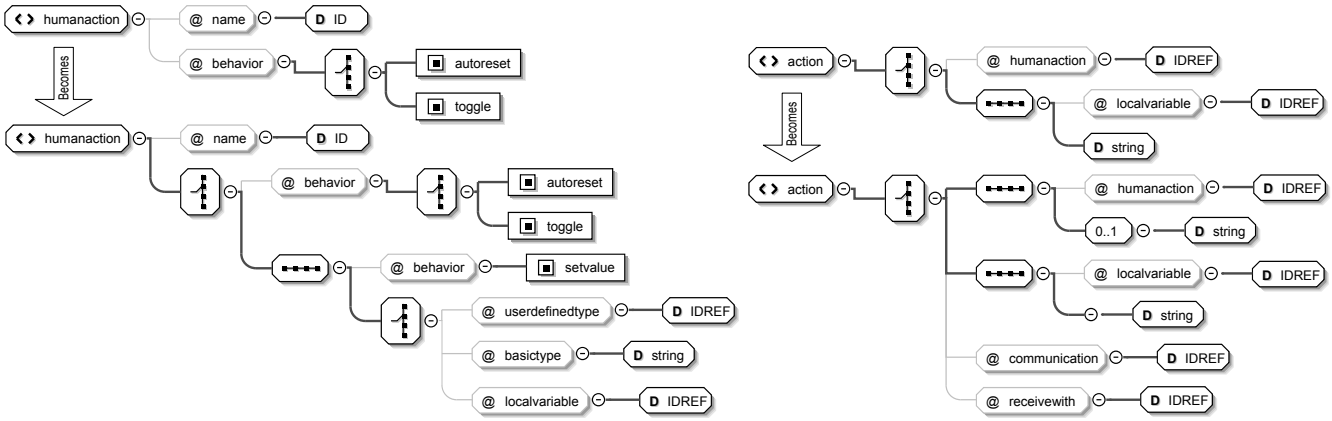

Fig. 3. Changes to the notation of the EOFM notation [5], [6] depicted in Relax NG schema diagram notation [17]. (a) Changes made to the *humanaction* declarations to allow for human action outputs that set specific values. (b) Changes to the definition of actions (which occur at the bottom of the task analytic model hierarchy) to accommodate the actual performance of actions for setting values, communicate information, and receive communications.

$$
\begin{array}{rll}
\mathit{userdefinedtype} ::= & (\mathit{id},\mathit{typedef}) & \mathit{type} ::= \quad \mathit{basictype} \mid \mathit{userdefinedtype} \\
\mathit{constant} ::= & (\mathit{id},\mathit{type},\mathit{val}) & \mathit{localvariable} ::= \quad (\mathit{id},\mathit{type}, \text{initialvalue: } \mathit{val}) \\
\mathit{inputvariable} ::= & (\mathit{id},\mathit{type}) & \mathit{behavior} ::= \quad \text{autoreset} \mid \text{toggle} \mid (\text{setvalue}, \mathit{type}) \\
\mathit{humanaction} ::= & (\mathit{id},\mathit{behavior}) & \\
\mathit{conditions} ::= & (\text{pre: } \mathit{condition}, \text{completion: } \mathit{condition}, \text{repeat: } \mathit{condition}) & \\
\mathit{activity\_operator} ::= & \text{or\_seq} \mid \text{or\_par} \mid \text{optor\_seq} \mid \text{optor\_par} \mid \text{and\_par} \mid \text{and\_seq} \mid \text{xor} \mid \text{ord} & \\
\mathit{action\_operator} ::= & \mathit{activity\_operator} \mid \text{sync} & \\
\mathit{action} ::= & \text{humanaction}(\mathit{id},\mathit{val}) \mid \text{localvariable}(\mathit{id},\mathit{val}) \mid \text{communication}(\mathit{val}) \mid & \\
& \text{receivewith}(\mathit{id}) & \\
\mathit{decomposition} ::= & (\mathit{activity\_operator},\mathit{activities}) \mid (\mathit{action\_operator},\mathit{actions}) & \\
\mathit{activity} ::= & (\mathit{id},\mathit{conditions},\mathit{decomposition}) & \\
\mathit{humanoperator} ::= & (\mathit{id},\mathit{inputvariables},\mathit{localvariables},\mathit{humanactions},\mathit{activities}) & \\
\mathit{eofms} ::= & (\mathit{constants},\mathit{userdefinedtypes},\mathit{humanoperators}) &
\end{array}
$$

Fig. 4. Abstract Syntax of EOFMC

$$
\frac{\text{start}(t[])}{\mathit{env}, t[] \vdash (\text{humanaction}(a,v), \text{Ready})) \xrightarrow{\{(a,v)\}} (\mathit{lenv}, (\text{humanaction}(a,v), \text{Executing}))}
$$

$$
\frac{\text{start}(t[])}{\begin{array}{c}\mathit{env}, t[] \vdash (\mathit{lenv}, (\text{localvariable}(x,v), \text{Ready})) \\ \longrightarrow (\mathit{lenv} + \{x \mapsto v\}, (\text{localvariable}(x,v), \text{Executing}))\end{array}}
$$

$$
\frac{\begin{array}{c}\text{start}(t[]),\ \forall j = 1,\ldots,k.\text{start}(t_{i_j}[]) \\ c = (n, \mathit{lenv}, [\ldots, t[(\text{communication}(v), \text{Ready})], \ldots]) \\ h_{i_j} = (n_{i_j}, \mathit{lenv}_{i_j}, [\ldots, t_{i_j}[(\text{receivewith}(x_{i_j}), \text{Ready})], \ldots]), j = 1,\ldots,k \\ c' = (n, \mathit{lenv}, [\ldots, t[(\text{communication}(v), \text{Executing})], \ldots]) \\ h'_{i_j} = (n_{i_j}, \mathit{lenv}_{i_j} + \{x_{i_j} \mapsto v\}, [\ldots, t_{i_j}[(\text{receivewith}(x_{i_j}), \text{Executing})], \ldots]), j = 1,\ldots,k\end{array}}{\mathit{env} \vdash [\ldots, c, \ldots, h_{i_1}, \ldots, h_{i_k}, \ldots] \longrightarrow [\ldots, c', \ldots, h'_{i_1}, \ldots, h'_{i_k}, \ldots]}
$$

$$
\frac{}{\mathit{env}, t[] \vdash (\mathit{lenv}, (\mathit{action}, \text{Executing})) \longrightarrow (\mathit{lenv}, (\mathit{action}, \text{Done}))}
$$

Fig. 5. Sample Transition Rules

monitoring and the pilot flying, leaving the Air Traffic Control (ATC) as part of the environment.

The second step is to attempt to model the behavior of the chosen operators using the target language (EOFMC). Since the language is, among other representations, encoded in an interactive theorem prover, we are able to check during the development process whether even an incomplete model satisfies its specification. Using the semantic model described in Section III, we can explore and extend the partial model until it satisfies all the specified properties. EOFMC allows us to easily refine our models to include more detail, by for instance replacing a single human action with an activity that accomplishes the same work at a finer level of detail.

B. Example

The visualization of EOFMC is nearly identical to that of EOFM [6]. The task hierarchy is displayed as a tree structure, in which each activity is connected to its sub-activities by an arc annotated with its decomposition operator. Incoming yellow arrows denote preconditions, outgoing pink arrows correspond to completion conditions, and loops denote repeat conditions, where the conditions themselves are shown on the arcs. Where a condition is not shown, it can be assumed to default to True.

Figures 6 and 7 show the main activities of the pilot flying and the pilot monitoring, respectively. aCommunicatePF models the behavior of the pilot flying when listening for the pilot monitoring's orders, while aPerformCDAPF models the actions taken in response to those orders during a CDA. aMonitorATCRestrictions models the behavior of the pilot monitoring during the approach to starting a CDA. Note that the activities in aPerformCDA correspond to the informal description in Section II of the pilot monitoring's initial tasks: requesting clearance, entering the clearance information into the Flight Management System (FMS), and directing the pilot flying to take action as needed. These task models are connected with an ord decomposition, ensuring that they will all be performed in the proper order. aMonitorCDA is the activity that corresponds to the pilot monitoring's actions over the course of the CDA, namely monitoring airspeed, making adjustments as needed, and aborting the CDA if no possible adjustments will keep the plane on track. As such, aMonitorCDA combines with an or_par decomposition the activity that monitors the speed and the one that checks for unrecoverable situations and aborts the CDA.

As discussed in Section II, in order to ensure adequate separation during a CDA, it is important that the pilots constantly manage the aircraft's speed. The actions taken for this purpose by the pilot monitoring are detailed in the aMonitorCDA activity. The activity has two main sub-activities, corresponding to the case where the aircraft's speed is off the target speed but correctable, and the case where the speed is off target beyond the pilots' ability to correct. Since the CDA should not be aborted unless absolutely necessary, these various monitoring activities are connected with an or_par operator, which allows the activity to complete even if one of its sub-activities has not. The use of the parallel modality here allows the pilot monitoring to signal an abort of the CDA as soon as he realizes that the CDA is impossible to complete, even if other activities are already in progress. Since the focus of our analysis is the CDA procedure, the fallback procedures that would be used after aborting the CDA, e.g., a go-around, are not modeled. Since the preconditions and completion conditions of the aAboveTargetSpeed and aBelowTargetSpeed activities are mutually exclusive, the pilot will not attempt to both reduce and increase the speed at once.

In the case that the aircraft's ground speed is too high by a correctable amount, and the aircraft's speed brakes are not deployed, the prescribed action is to deploy the speed brakes. The pilot monitoring, having recognized this situation, performs the subactivity of aAboveTargetSpeed, titled aDeploySpeedBrakesM. In this activity, the pilot monitoring tells the pilot flying to deploy the speed brakes; simultaneously, in the aCommunicatePF activity, this order is stored in the pilot flying's local variable lOrderPF. At this point, the pilot flying is expected to be in the loop of the aPerformCDAPF activity, and thus upon hearing the order will deploy the speed brakes (activity aDeploySpeedBrakes1PF), after ensuring safety via the precondition iIAS $<$ cSpeedBrakeOverspeedLimit. iIAS is the input variable that indicates the aircraft's airspeed, so this check ensures that brakes are not deployed in a situation in which they would be damaged. The other orders of the pilot monitoring are similarly passed along and checked for safety by the pilot flying. In this way, the extension of EOFM with communication allows us to model a CDA in which the pilot flying and the pilot monitoring work cooperatively to ensure a common goal.

V. Checking Properties

For the ATC to maintain separation between the multiple incoming aircraft he must be able to predict roughly the location of the aircraft under his management. In order for the ATC to maintain his safety guarantees he relies on the flight crew to, among other things, keep their speed close to their assigned speed during a CDA. Thus, as part of the verification of a CDA task behavior for pilots, we would like to show that the aircraft's speed is kept within a certain margin, or that the CDA is aborted if the speed goes outside that margin. Such a proof would involve several components, including a model of the environment (wind, mechanical failures, etc.) and the effects of the pilots' actions on the aircraft's speed. Our work so far has focused on the components described in EOFMC: the pilots, and the portion of the aircraft equipment under their control.

Using a formalization of EOFMC syntax and semantics in the Isabelle theorem prover [18], we have taken the first step towards verifying the CDA procedure. We consider the situation in which the aircraft's speed exceeds the desired speed, but remains within the safety margin. In this situation, the response prescribed by CDA is to deploy speed brakes to ensure that the speed remains within the margin. We aim to show that the EOFMC specification of the CDA procedure does in fact guarantee that if such a situation arises, the speed brakes will eventually be deployed.

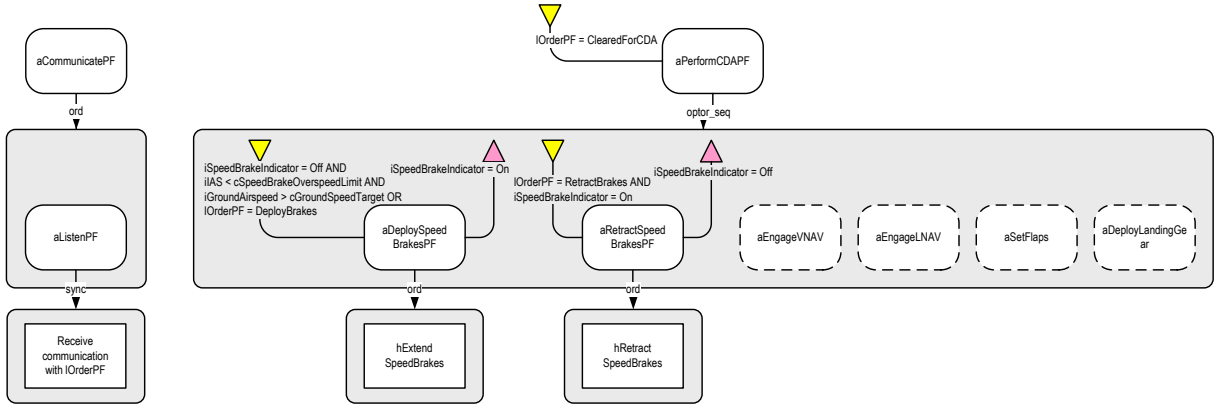

Fig. 6. Pilot Flying CDA description (activities with dotted lines not fully depicted)

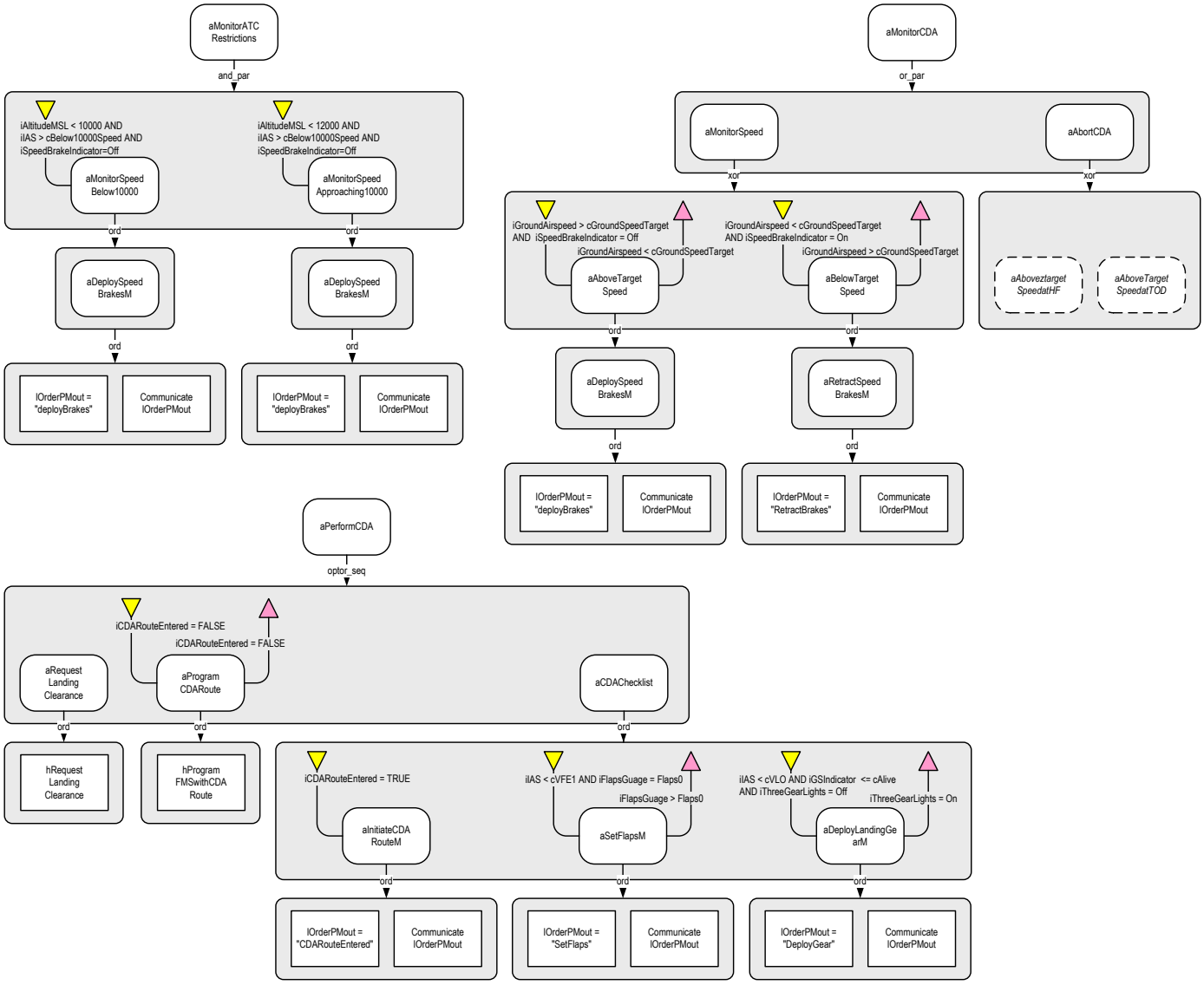

Fig. 7. Pilot Monitoring CDA description (activities with dotted lines not fully depicted)

The deployment of the speed brakes in the task specification can be broken down into three steps. First, the Pilot Monitoring must observe that the aircraft's airspeed has exceeded a certain level, which may vary between stages of the CDA process. After making this observation, the Pilot Monitoring gives the order to deploy the speed brakes. The Pilot Flying, in response to this order, checks that it is safe to carry out (i.e., that the aircraft's speed does not exceed the overspeed limit for the speed brakes), and performs the physical action necessary to deploy the brakes.

Specifications of human behavior are particularly prone to problems of livelock – a process may be allowed to repeat indefinitely, limited only by the agent's discretion. As such, in order to verify properties other than safety properties (e.g., that some event does eventually occur), we require an assumption of *fairness*. Our definition of fairness is analogous to that used in LTL and other temporal logics: if a transition is possible infinitely often, then it will eventually be taken [19]. Under this assumption, we can show that if the pilot monitoring orders the pilot flying to deploy the speed brakes (is ready to perform the communicate("deploy brakes") action), the pilot flying will eventually hear the order. Similarly, if the pilot flying receives the "deploy brakes" order, then under the fairness assumption the speed brakes will eventually be deployed (the hExtendSpeedBrakes action will be performed), as long as the aircraft's speed is not so high as to risk damage to the brakes.

VI. Conclusions and Future Work

Thus far, we have extended the syntax of EOFM to allow for communication and transmission of values, and given semantics for the extended language EOFMC. An EOFMC specification can also be translated into XML, which can then serve as a top-level model for other analysis tools. Using this language, we modeled the behavior of two of the human operators in the CDA procedure, and sketched a proof of a property that would help the ATC ensure the safety of the procedure. Further proofs along these lines could help guarantee the safety of a CDA task model, or find errors in the definition of the procedure.

However, the techniques described so far do not account for a range of unanticipated errors, such as hardware failures and miscommunication on the part of the humans. Rather than attempting to provide absolute guarantees against unlikely but catastrophic errors, e.g. multiple engine failure, we might use a probabilistic view of the environment to show that we have a high probability of safety in all situations.

Synchronous communication is only a rough approximation of the complexities of information transfer between humans. In real-world situations, messages can be ignored, lost, misspoken, or misheard. We would like to use a communication primitive that captures these imperfections, and is sufficiently flexible to model both synchronous and asynchronous communication. By extending EOFMC's communication model with such a primitive, we will be able to deal with an increased range of potential errors.

When writing the initial specification for a scenario, the writer may not know all the details of the activities involved. Conversely, when analyzing a specification, we may find it appropriate to increase or decrease the level of detail according to the property under consideration. To this end, it is our goal to extend EOFM with an abstraction mechanism, by which an action can be expanded into a full activity, or an activity collapsed into a single action. This must be done in a manner that preserves the properties under consideration, and gives rise to various complications regarding concurrency and simultaneity. Nonetheless, we believe that it would be a considerable aid in using EOFMC to verify specifications of complex behavior.

VII. Related Work

While there is a large body of work on modeling human-automation interaction, several topics are particularly relevant to our work with EOFMC. Input/output automata [20] are similar to our task tree models in that they are expressed as transition systems with input and output variables. However, EOFMC gives control over the composition of sub-activities as a first-class feature of the language, via its decomposition operators. UML [21] is a visualization scheme analogous to that used for EOFM, but is not specific to human task modeling. ConcurTaskTrees [22], like EOFMC models, have a hierarchy of tasks, but only express systems with a single human operator, and thus do not handle human-human communication.

VIII. Acknowledgements*

This work was supported in part by NASA Contract NNA10DE79C and NSF Grant 0917218. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NASA or NSF.

References

[1] BASI, "Advanced technology aircraft safety survey repor," Department of Transport and Regional Dedvelopment, Bureau of Air Safety Investigation, Civic Square, Tech. Rep., 1998.

[2] FAA Human Factors Team, "Federal aviation administration human factors team report on: The interfaces between flightcrews and modern flight deck systems," Federal Aviation Administration, Washington, DC, Tech. Rep., 1996.

[3] D. Hughes and M. A. Dornheim, "Accidents direct focus on cockpit automation," *Aviation Week and Space Technology*, vol. 142, no. 5, pp. 52–54, 1995.

[4] J. B. Sexton and R. L. Helmreich, "Analyzing cockpit communications: the links between language, performance, error, and workload," in *Hum Perf Extrem Environ*, vol. 5, no. 1, Oct. 2000, pp. 63–68.

[5] M. L. Bolton and E. J. Bass, "Enhanced operator function model: A generic human task behavior modeling language," in *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*. Piscataway: IEEE, 2009, pp. 2983–2990.

[6] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass, "A systematic approach to model checking human-automation interaction using task-analytic models," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 2010, in Press.

[7] ""B-757/767 Constant Descent Approach (CDA) Procedures"," *General Information Bulletin*.

[8] J. Brooks, "Continuous descent arrivals," in *ICAO Workshop on Aviation Operational Measures for Fuel and Emissions Reductions*, 2006.

[9] J.-P. B. Clarke, N. T. Ho, and L. Ren, "Continuous descent approach: Design and flight test for Louisville International Airport," *Journal of Aircraft*, vol. 41, no. 5, pp. 1054–1066, 2004.

[10] J.-P. B. Clarke and L. Ren, "Flight demonstration of the separation analysis methodology for continuous descent arrival," 2007, pp. 1–10.
[11] J.-P. Clarke, "Continuous descent arrivals: Noise, emissions, and fuel burn reductions," 2007.
[12] H. J. Reynolds, T. G. Reynolds, and R. J. Hansman, "Human factors implications of continuous descent approach procedures for noise abatement in air traffic control," in *6^{th} USA/Europe Air Traffic Management R&D Seminar*, 2005.
[13] C. R. Spitzer, "Flight management systems," in *The Avionics Handbook*. Boca Raton: CRC, 2001, pp. 1501–1525.
[14] W. White, "Southern california TRACON CDA design overview: How does CDA fit?" in *CDA Workshop Presentation*, 2006.
[15] ——, "Southern california TRACON: Optimized descent," in *CDA Workshop Presentation*, 2006.
[16] "SAL home page." [Online]. Available: http://sal.csl.sri.com/
[17] "Relax NG schema diagram," Syncro Soft. [Online]. Available: http://www.oxygenxml.com/doc/ug-oxygen/topics/relax-ng-schema-diagram.html
[18] L. C. Paulson, "Isabelle: The next 700 theorem provers," in *Logic and Computer Science*, P. Odifreddi, Ed. Academic Press, 1990, pp. 361–386.
[19] D. A. Peled, *Software reliability methods*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.
[20] N. Lynch and M. Tuttle, "An introduction to input/output automata," *CWI-Quarterly*.
[21] J. Rumbaugh, I. Jacobson, and G. Booch, Eds., *The Unified Modeling Language reference manual*. Essex, UK, UK: Addison-Wesley Longman Ltd., 1999.
[22] F. Paterno, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models." Chapman & Hall, 1997, pp. 362–369.