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

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

*Department of Systems and Information Engineering, University of Virginia, Charlottesville, VA 22904, {ejb4n,mlb4b}@virginia.edu
†School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30313, kfeigh@isye.gatech.edu
‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, {dgriffi3,egunter,mansky1}@illinois.edu
§Computer Science Laboratory, SRI International, Menlo Park, CA 94025, rushby@csl.sri.com

***Abstract*—Breakdowns in complex systems often occur as a result of system elements interacting in ways unanticipated by analysts or designers. 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 future support for model checking linked through the EOFMC top level XML description.**

Herein, we consider an aviation scenario in which an air traffic controller needs a flight crew to change the heading for spacing. Although this example, at first glance, seems to be one simple task, on closer inspection we find that it involves local human-human communication, remote human-human communication, multi-party communications, communication protocols, and human-automation interaction. We show how all these varied communications can be handled within the context of EOFMC.

***Index Terms*—Task analysis, human-automation interaction, human-computer interaction, formal verification, theorem proving, model checking**

I. Introduction

Failures in safety-critical systems can arise due to interactions between system elements, including human operators. 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 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 as the potential ramifications of human behaviors can be verified in relation to other aspects of the system.

The Enhanced Operator Function Model (EOFM) [5] is one method of specifying human behavior in a way amenable to formal analysis. In this paper, we are extending EOFM as part of a multi-method approach where analyses via theorem proving and model checking are linked through a top-level XML description of human task behavior. Our model of cooperative activity involves two or more humans and their automated systems functioning within specified roles. When collocated, human agents can communicate with each other verbally and through gesturing, and when not, through the use of phone, radio or video communications as well as text data (electronic mail, and text and instant messaging). Human operators perform activities manually and through the use of automated devices. Cooperative activity may involve individual agents completing activities asynchronously or synchronously. They may complete certain activities interleaved in a coordinated manner. The order of activities may be pre-specified or may depend upon the status of the activities or other system components or events.

Herein we consider an example from aviation where the human agents are an air traffic controller (ATCo) and the two pilots of an aircraft with modern automation, the pilot flying (PF) and the pilot monitoring (PM). This example displays the need to handle different types of communication (verbal and gestures) within a single modeling framework. We focus on human communications, on pilot interaction with other pilots and with automation, and on the properties needed to be guaranteed by the pilots to ensure that the aircraft is always compliant with clearances. The following sections describe our new formalisms, provide a heading example, sketch the proof showing how safety properties can be guaranteed, and discuss future and related work.

II. EOFM with Communication (EOFMC)

A. Overview and Syntax

We have developed a variant of EOFM, called EOFMC (EOFM with Communication). EOFM is an XML-based, platform- and analysis-independent language for describing task analytic models [5]. 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

978-1-4577-0653-0/11/$26.00

(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 operators, automation, or other system elements.

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. Decomposition operators specify the cardinality of and temporal relationship between the sub-activities or actions:

- or_seq, or_par – 1 or more sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- optor_seq, optor_par – 0 or more sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- and_seq, and_par – all sub-activities or actions must execute either one at a time or with any possible overlap respectively;
- sync – all actions (activities are not allowed in sync decompositions) must execute at the same, synchronized time;
- xor – exactly 1 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 conditions (Boolean expressions 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). Each EOFM atomic action is 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 controlling how tasks execute. action nodes are leaves in the hierarchy and reference humanaction nodes (indicating that a human action is performed) or allow values to be assigned to local variables.

The heading change analysis is based on EOFMC, an extension of EOFM to support human actions that set specific values, activities shared across multiple human operators and human-human communications. Three modifications were made to existing EOFM constructs (Fig. 1). The syntax for constant nodes was updated to allow for the definition of constant mappings and functions through the addition of optional (zero or more) parameter attributes for defining function arguments.

The notation for defining human action outputs (humanaction nodes) now allows for an additional type of behavior called setvalue. Originally, human actions were treated as binary (either being performed or not being performed). They could exhibit two behaviors: autoreset, where the action would be performed and then automatically transition to not being performed; and toggle, where the execution of the action would toggle between being performed and not being performed. 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 (a reference to a userdefinedtype or a basictype) or a reference to a local variable which will contain the value to be set.

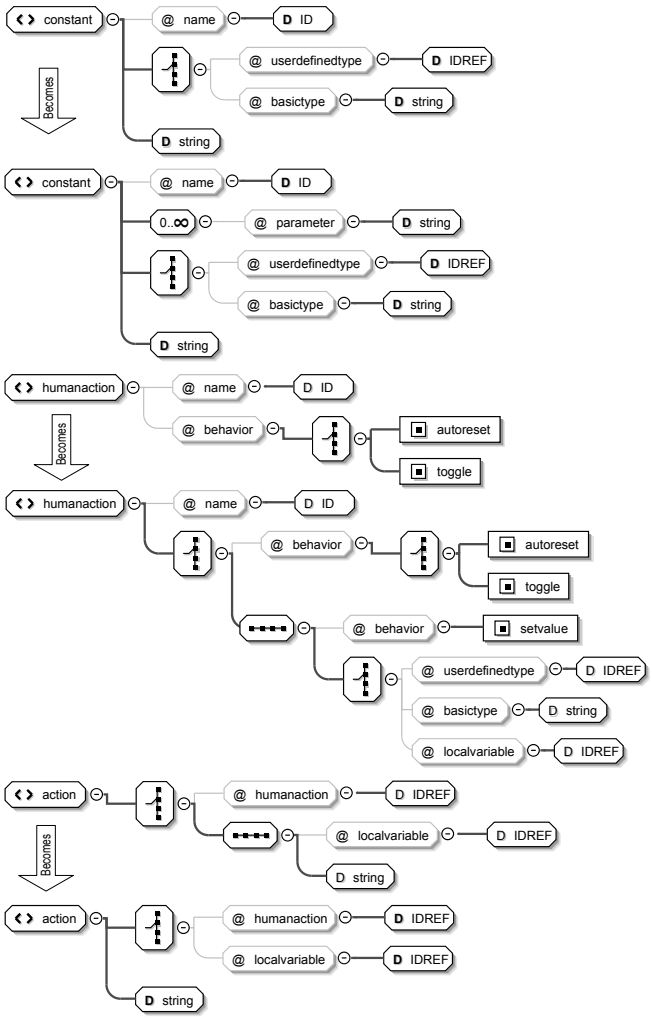

Fig. 1. Changes made to the notation of EOFM notation [5] depicted in Relax NG schema diagram notation [6]. Changes made to the constant nodes allow for the definition of parameters for function and mapping declarations. humanaction nodes have been updated to allow for human action outputs that set specific values. Changes to action nodes (which occur at the bottom of the task analytic model hierarchy) allow for the setting of values.

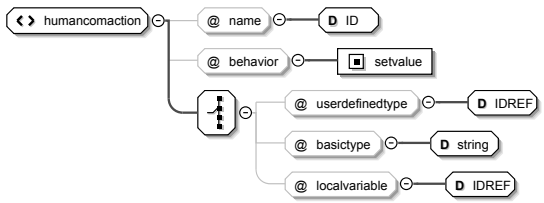

Fig. 2. The Relax NG schema diagram notation [6] for the new human communication action (humancomaction).

The third change impacts how actions are specified (action nodes). In base EOFM, 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).

Two additions were also made to the EOFM notation. The first added a new type of human action, humancomaction, for supporting human-human communication actions (Fig. 2). This is a special type of human action that is defined in the same way as a setvalue human action with the presumption that, when instantiated, each humancomaction represents a single communication modality. They are defined at the same location as humanaction nodes (Fig. 3).

EOFMC introduces the idea of shared tasks, which represent coordinated group activities undertaken by two or more human operators while allowing for human to human communication. These shared tasks are optionally defined after humanoperators in sharedeofm nodes (Fig. 3). Each sharedeofm contains two or more associate nodes explicitly defining the participating human operators. Tasks themselves are represented with the same hierarchy of activities and actions as in individual human operator tasks.

There are however, two differences. Firstly, in each activity, one can define zero or more associate nodes to further restrict which human operators are involved. Each action node belongs to an individual human operator and does not require an explicitly defined associate[1]. The second difference relates to activity decomposition. While the activities in these shared tasks can decompose in the same ways as their single-operator counterparts, they have an additional decomposition option. This new com decomposition operator, shown in Fig. 3, explicitly models human-human communication. It is a special form of the sync decomposition operator and assumes information is being transferred between human operators. The decomposition must start with the performance of a humancomaction, which commits a value (communicated information). The decomposition ends with one or more actions explicitly pointing to local variables to allow other human operators to register the information communicated via the humancomaction.

[1]The associates of each activity or action must be a subset of the associates of its ancestors. If no associates are defined in an activity, the associates are inherited from its parent activity.

We have developed an XML-based syntax for EOFMC that extends the syntax of EOFM, as well as a RelaxNG specification that accepts expressions conforming to this grammar and checks that all variables are declared before use. The formal semantics of EOFMC allow automated translation into the SAL model checker [5], [7]. 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. 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 called *task trees* to model the semantics of tasks. In a task tree, internal nodes correspond to the specifications of activities and leaves correspond to atomic actions. Each node in the tree is additionally labeled with one of three states: Ready, Executing, and Done. During execution of a task, these labels are updated. 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 [5], with one slight revision: an activity cannot transition from Ready to Done unless its start and end conditions are both satisfied. In order to compute these implicit conditions, we associate with each activity an accompanying *context*, which has the structure of the top-level task tree under consideration with a *hole* at the location of the activity. These contexts will be carried along in any execution.

Given an environment $lenv$ supplying values for the local variables, a sequence of global environments $envs$ supplying the history of values for input and output variables, a function val giving values for expressions (using $lenv$ and $\mathsf{hd}(envs)$, the most recent global environment in $envs$), and a function sys mapping $envs$ to a new global environment env representing the system's response to the human actions, a complete EOFMC specification is modeled by a set of human operators $[h_1, \ldots, h_m]$, a set of task trees for shared activities $[s_1, \ldots, s_n]$ and a set of rules for how they may transform. Each human operator has a name n and a collection of task trees $[t_1, \ldots, t_l]$, one for each top-level activity in the operator's specification.

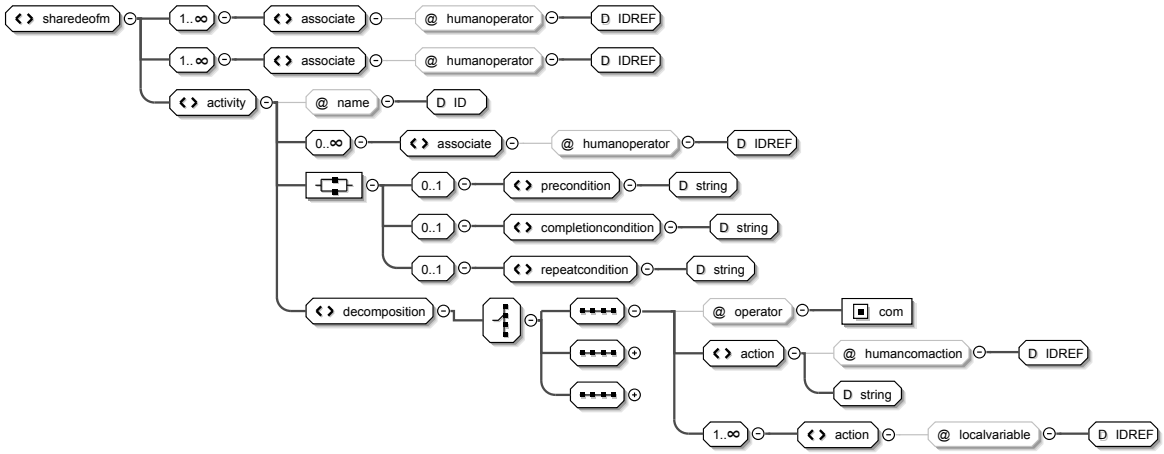

Fig. 3. Changes to the notation of the EOFM notation depicted in Relax NG schema diagram notation. Changes to the humanaction declarations allow for human action outputs that set specific values. Changes to the definition of actions (at the bottom of the task analytic model hierarchy) accommodate actions that set values, communicate information, and receive communications. The new com decomposition operator indicates information sharing between human operators.

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 zero or more activities may advance, and zero 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 task tree, which come in three basic types: HumanAction corresponding to a humanaction for an action observable by the system and other humans, LocVar corresponding to localvariable for remembering values, and Com for the combined communication action of a com decomposition in a sharedeofm.

The transformation of environments and task trees is governed by a set of transition rules for these actions. Each transition rule assumes a function sys modeling the system with which the human operators are interacting, and a context ("task tree with a hole") $t[]$ giving the task tree surrounding the action in question. As shown by the first rule, a human action $\mathsf{HumanAction}(a, v)$ causes the operator to perform operation a with value w (computed from v in $lenv$, the local environment, and $\mathsf{hd}(envs)$, the current global environment), causing the environment to change the value of the output variable a in the environment to w, and causing the action to transition to the Executing state. This can occur as long as its context $t[]$ satisfies the *start* condition at the hole and the action itself is in the Ready state. In the second rule, a local variable action $\mathsf{LocVar}(x, v)$ causes the operator to associate the value of v with the variable x in the human's local environment, modeling the action of remembering or making a note of a value. The behavior of the third, combined action $\mathsf{Com}(n, v, x_1, \ldots, x_m)$ causes each of the local variables (belonging to various humans) to be updated with the value of v (set by the human operator named n). The fourth rule tells us that any action that is Executing may immediately transition to Done; an action's

$$\frac{\mathsf{start}(t[]) \qquad \mathsf{val}_{lenv,\mathsf{hd}(envs)}(v) = w}{sys, t[] \vdash (envs, lenv, (\mathsf{HumanAction}(a, v), \mathsf{Ready})) \xrightarrow{\{(a,w)\}} (\mathsf{hd}(envs) + \{a \mapsto w\} @ envs, lenv, (\mathsf{HumanAction}(a, v), \mathsf{Executing}))}$$

$$\frac{\mathsf{start}(t[]) \qquad \mathsf{val}_{lenv,\mathsf{hd}(envs)}(v) = w}{sys, t[] \vdash (envs, lenv, (\mathsf{LocVar}(x, v), \mathsf{Ready})) \longrightarrow (envs, lenv + \{x \mapsto w\}, (\mathsf{LocVar}(x, v), \mathsf{Executing}))}$$

$$\frac{\mathsf{start}(t[]) \qquad \mathsf{val}_{lenv,\mathsf{hd}(envs)}(v) = w}{sys, t[] \vdash (envs, lenv, (\mathsf{Com}(n, v, x_1, \ldots, x_m), \mathsf{Ready})) \longrightarrow (envs, lenv + \{x_1, \ldots, x_m \mapsto w\} (\mathsf{Com}(n, v, x_1, \ldots, x_m), \mathsf{Executing}))}$$

$$\frac{}{sys, t[] \vdash (envs, lenv, (action, \mathsf{Executing})) \longrightarrow (envs, lenv, (action, \mathsf{Done}))}$$

$$\frac{}{sys, t[] \vdash (envs, lenv, (action, state)) \longrightarrow (sys(envs) @ envs, lenv, (action, state))}$$

Fig. 4. Sample Transition Rules

end condition is always true by definition. The last rule says that the system may at any time act by updating the environment (using information from the global environment), but with no effect on the task trees. Using these rules, and more for activities and *eofms*, we can give EOFMC specifications a transition semantics specifying the steps of execution.

An *execution* of a specification is defined as a series of 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. Given an EOFMC specification of a task behavior, we can use this model to prove safety properties of

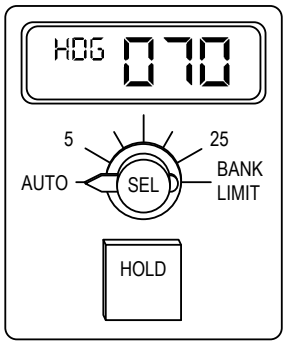

Fig. 5. Mode Control Panel Heading Control and Display

the task behavior. In addition, there is a precise relationship between our semantics and the translation to SAL given for earlier versions of EOFM [5]. 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 the semantics for EOFM.

III. An Example Model

In order to illustrate how our approach can be used to model human-human and human-automation interaction, we construct an instantiated EOFMC model for a heading change example with three human operators, the two pilots (PF and PM) and the air traffic controller (ATCo) (Fig. 6). This shared activity starts when, for spacing, the ATCo wants to clear the aircraft to a new heading. The ATCo's clearance is a verbal communication that both pilots hear. The two pilots collaboratively implement the heading change. The PF monitors the heading change using a combination of cues; for instance, he checks that the heading is changing appropriately on the horizontal situation indicator (HSI) and that the HDG SEL mode is active. The ATCo monitors that the aircraft has executed the heading change by noting the change in its track.

With respect to the aircraft, the Autopilot Flight Director System consists of Flight Control Computers and the Mode Control Panel (MCP). The MCP provides control of the Autopilot (A/P), Flight Director, and the Autothrottle System. When the A/P is engaged, the MCP sends commands to the aircraft pitch and roll servos to operate the aircraft flight control surfaces. Herein the MCP is used to activate heading changes. The Heading (HDG)/Tracking (TRK) window of the MCP displays the selected heading or track (Fig. 5). The 3 digit numeric display provides the current desired heading in compass degrees (between 0 and 359). Below the HDG window is the heading select knob. Changes in the heading are achieved by rotating and pulling the knob. Pulling the knob tells the autopilot to use the pilot selected value and engages the HDG mode.

The ATCo initiates the heading clearance using radio communications. The ATCo and the PM communicate over the radio with a pre-specified communication protocol. The heading is communicated and confirmed in a sequential process. First, the ATCo presses his push-to-talk switch. Then, the ATCo communicates the heading (*lATCoSelectedClearance*) to the pilots (via the *hATCoTalk* humancomaction), such as "AC1 Heading 070 for spacing". Both pilots remember this heading (stored in the local variables *lPFHeadingFromATC* and *lPMHeadingFromATC* for the PF and PM respectively). The ATCo releases the switch. Next, the PM presses his switch. The PM then repeats/communicates the heading that he heard (in this example, "AC1 Heading 070"), where both the ATCo and PF hear and remember the heading. The PM releases the switch. This entire process must repeat if the heading the ATCo hears from the PM does not match the heading he wanted to communicate (*lATCoSelectedClearance* $\neq$ *lATCoHeadingHeardFromPilots*). It completes otherwise.

Once the heading has been communicated, the pilots collaborate (*aSetNewHeading*). This process involves selecting, confirming (*aChangeAndConfirm*) and then executing the new heading (*aExecuteTheChange*). The selection and confirmation process starts with the PF pushing and rotating the heading select knob to the perceived heading (to 70 degrees in this example)[2]. The PF pulls the knob. The PM verifies that the PF has dialed the correct heading and confirms the heading selection by pointing to the heading selection in the window and stating "Heading 070". Here two communications occur in parallel (indicated by the and_par decomposition operator associated with *aConfirmTheChange*): the PM points at the heading window (*aPointAtHeadingWindow*) and he speaks the heading that was entered (*aSayTheHeading*). Both are perceived by the PF. This process must repeat if the heading perceived by the PF does not match the heading spoken by the PM (*lPFHeadingFromPM* $\neq$ *lPFHeadingFromATCo*). Once the heading is confirmed, the PF then presses the heading select button to execute the heading change.

IV. Checking Properties

In the example protocol discussed in this paper, there are certain basic properties we must have. Most importantly, we want to know that the pilots' setting the aircraft to a new heading will only occur as the result of the pilots' having been given clearance by the air traffic controller for that heading and not having been given clearance for any other heading since then. This says the pilots will not arbitrarily change the heading of the aircraft, a basic safety property (a safety property guarantees that some undesirable event will never happen). Perhaps just as important, however, is the property that the air traffic controller's giving clearance to the pilots to change to a new heading will lead to the pilots' changing the aircraft to that new heading. This property is a progress property (one that guarantees that some desirable event will eventually happen), rather than a safety property. It is arguable that the second property is as important as the first, however, since the air traffic controller needs to know he can rely on his directives being followed.

To be able to analyze these in the interactive theorem prover Isabelle, we translated the XML rendition of the EOFMC

[2]The heading is converted to a numeric heading value using the *cSayableToHeading* constant mapping. This mapping converts communications encompassed by *hATCoTalk*. *cHeadingToSayable* converts the pilot communications.

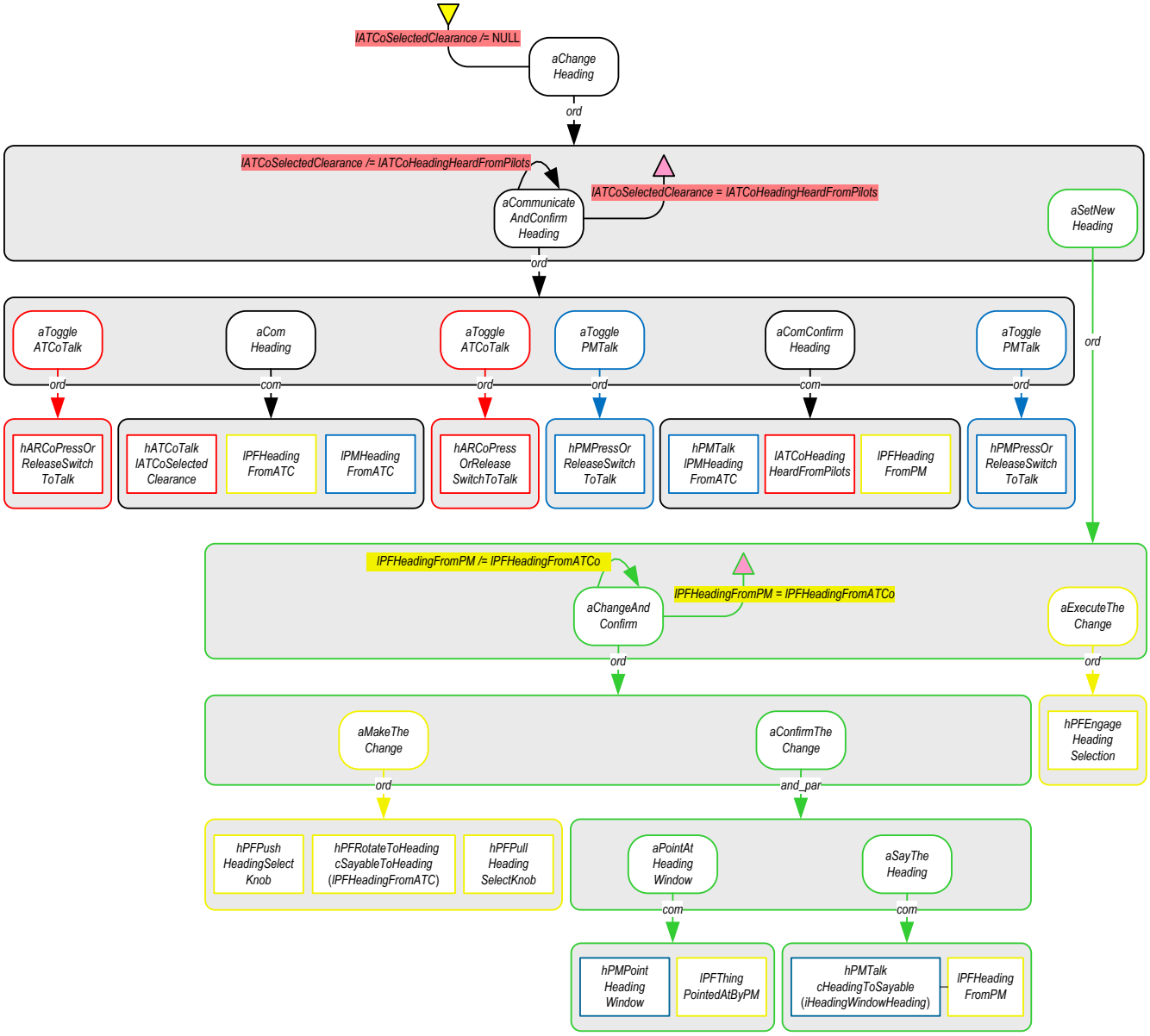

Fig. 6. Visualization of the instantiated EOFMC shared task for changing the heading on an aircraft. The visualization of EOFMC is nearly identical to that of EOFM [5]. The task hierarchy is displayed as a tree structure in which each activity is connected to its sub-acts by an arc annotated with its decomposition operator. Actions are rectangles and activities are rounded rectangles. Connected yellow triangles represent preconditions, magenta triangles indicate completion conditions, and loops denote repeat conditions. Condition logic annotates the arcs. Where a condition is not shown, it can be assumed to default to true. Activity names are prefixed with '*a*', human actions with '*h*', input variables with '*i*', local variables with '*l*', and constants with '*c*'. Activities, actions, and condition logic are color coded based on the human operator associated with them: red for the ATCo, yellow for the PF, blue for the PM, green for the PF and PM, and *black* for all three.

specification shown in Fig. 6 into our internal version in Isabelle. This translation was done by hand, but it was done systematically by recursion over the structure.

The properties given above are stated in Isabelle as properties of lists of task trees and accompanying environment information. These lists need to conform to execution sequences, and thus the head element of the list must correspond to the initial task tree derived from the syntax tree representing the EOFMC specification, and each adjacent pair should be related by the transition relation outlined in Section II-B. The method for showing such safety properties is induction on the length of the sequence, but requires strengthening the inductive hypothesis to track each step of information flow through the execution.

Such proofs, however, are a bit unsatisfying. The difficulty is that the semantics given to the protocol never allows the possibility of a faulty communication. As a result, there is no value other than the correct one to which the PF could set the heading. To address this, we need to add an additional transition to our system for communication where some of the local variables receive a different value from that transmitted, and consider sequences under this extended system. If we do so, freely allowing for the possibility of any number of miscommunications, then, in fact, we will find that even the safety property does not hold. Consider the situation where, when the ATCo says a heading of 050, both the PM and the PF hear 090, but when the PM repeats the heading, the ATCo hears 050 as he expected. In this case, the PF will go ahead and incorrectly change the heading to 090 with the approval of the PM.

The above scenario may seem unlikely, as it relies upon not only two temporally distinct communications going badly, but also their doing so in a very particular way with perfect coincidence in the nature of the miscommunication. To address this criticism, we investigated the weaker property of robustness against a single error in communication. Such a proof would use roughly the same proof outline as for safety, but with an almost overwhelming increase in the case analysis. Still, in the process of considering a proof of this property, we uncovered that, in fact, it too does not hold. The single point of failure occurs if, when the ATCo announces the new heading clearance, the PM hears the heading change correctly, but the PF does not. Because the PM has heard the correct result (and all subsequent communications are assumed not faulty), the protocol will move past the phase *aCommunicateAndConfirmHeading* and on to *aSetNewHeading*. This is the first point where there is a missed opportunity to avoid the error: there is no provision for the PF to take any action in the case that $lPFHeadingFromPM \neq lPFHeadingFromATCo$. The next phase of the protocol, *aChangeAndConfirm*, is designed to ensure that the value seen by the PM agrees with what the PF thought he had heard the ATCo say. Here, a second opportunity to detect the problem is missed, since the protocol does not allow the PM in *aConfirmTheChange* to notice a mismatch between the value the PF has dialed into the heading window and the value the PM heard from the ATCo.

The progress property is even more complex to deal with, in part because it can only possibly hold in the presence of some form of "fairness" assumption, which would require that every party has a chance to perform its actions. If the system is allowed to transition repeatedly without intervening steps of execution of the task tree, the world could go on forever without reaching the desired state where the PF engages the selection (or even sets it). Under a sufficient fairness condition, it should be possible to show the progress property for this particular protocol by first showing that every sequence that ends in a configuration where all the nodes in the task tree are marked Done had to pass through a transition where engaging the heading selection was executed, and second, showing that every sequence can be completed to one in which all the nodes in the task tree are labeled Done. The latter result would be shown by induction first on the number of nodes not marked Done, and second on the number of nodes marked Ready, and would rely upon the fundamentally sequential nature of the protocol.

V. 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 the human operators in the procedure to effect a change of heading clearance, and sketched a proof of a basic safety property. Further proofs along these lines could help guarantee the safety of the protocol's task model, or find errors in the definition of the procedure.

In this paper, we have addressed the issue of modeling human-human communication, and begun to address the issue of faulty communication (another area for future work). However, the techniques described so far do not account for a range of unanticipated errors, such as hardware failures. 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.

When writing the initial specification for a scenario, the analyst 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.

VI. Related Work

A variety of research has investigated how task analytic behavior models can be used with formal verification to evaluate

system issues related to human-automation interaction (see [8] for a deeper survey). Some of this work utilized general formal modeling notations such as state charts [9], Interacting Communicating Objects (ICOs) [10], and Communicating Sequential Processes (CSP) [11], in which it is the job of the analyst to represent task analytic modeling concepts in the formal notation. While such notations are very expressive, and the analyses using them can be very powerful, human factors and human-automation interaction engineers do not generally use such notations to represent human behavior. Rather, they use hierarchical task analytic modeling notations similar to EOFMC such as EOFM [5], ConcurTaskTrees (CTT) [12], and User Action Notation (UAN) [13]. Thus, in order to allow native task analytic models to be used in formal verification analyses, researchers have shown that models instantiated in these notations can be automatically translated into the more general formal modeling languages [5], [14], [15], [16].

With the exception of Paternò et al. [15], all of these techniques have focused on single operator systems. Paternò and colleagues identified three different ways that collaborative human behavior could be incorporated into formal task analytic behavior models (in this case CTT): (1) A single task model could be created describing the behavior over every operator in the system; (2) Separate task models could be created for each operator with additional operators (similar to decomposition operators in EOFMC) being used to describe the coordinated and communicative relationships between activities and action between operator tasks; and (3) A separate task model could be created for each operator but with the addition of a shared structure which relates the coordinated and communicative relationships between the activities and actions contained in the separate models. The work presented here offers a fourth means of accomplishing this: allowing separate task to be created for each operator for activities not requiring coordination, and allowing for separate shared tasks/activities that require multi-operator coordination and communication. It is our presumption that this approach will be more intuitive for human factors engineers because it does not require them to coordinate the execution of multiple task structures. Future work should investigate if this is indeed the case as compared to the other three approaches discussed by Paternò et al. [15].

REFERENCES

[1] BASI, "Advanced technology aircraft safety survey report," 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, 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*, 2011, in Press.

[6] "Relax NG schema diagram," Syncro Soft. [Online]. Available: http://www.oxygenxml.com/doc/ug-oxygen/topics/relax-ng-schema-diagram.html

[7] "SAL home page." [Online]. Available: http://sal.csl.sri.com/

[8] M. Bolton, E. J. Bass, and R. Siminiceanu, "Using formal verification to evaluate human-automation interaction, a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, in revision.

[9] A. Degani, M. Heymann, and M. Shafto, "Formal aspects of procedures: The problem of sequential correctness," in *Proceedings of the 43rd Annual Meeting of the Human Factors and Ergonomics Society*. Santa Monica: HFES, 1999, pp. 1113–1117.

[10] S. Basnyat, P. Palanque, B. Schupp, and P. Wright, "Formal socio-technical barrier modelling for safety-critical interactive systems design," *Safety Science*, vol. 45, no. 5, pp. 545–565, 2007.

[11] E. L. Gunter, A. Yasmeen, C. A. Gunter, and A. Nguyen, "Specifying and analyzing workflows for automated identification and data capture," in *Proceedings of the 42nd Hawaii International Conference on System Sciences*. Los Alatimos: IEEE Computer Society, 2009, pp. 1–11.

[12] F. Paterno, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models." Chapman & Hall, 1997, pp. 362–369.

[13] H. R. Hartson, A. C. Siochi, and D. Hix, "The UAN: A user-oriented representation for direct manipulation interface designs," *ACM Transactions on Information Systems*, vol. 8, no. 3, pp. 181–203, 1990.

[14] Y. Aït-Ameur and M. Baron, "Formal and experimental validation approaches in HCI systems design based on a shared event B model," *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 547–563, 2006.

[15] F. Paternò, C. Santoro, and S. Tahmassebi, "Formal model for cooperative tasks: Concepts and an application for en-route air traffic control," in *Proceedings of the 5th International Conference on Design, Specification, and Verification of Interactive Systems*. Vienna: Springer, 1998, pp. 71–86.

[16] R. E. Fields, "Analysis of erroneous actions in the design of critical systems," Ph.D. dissertation, University of York, York, 2001.