Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Enhanced Operator Function Model:

A Generic Human Task Behavior Modeling Language

Matthew L. Bolton and Ellen J. Bass
Department of Systems and Information Engineering
University of Virginia
Charlottesville, Virginia, United States of America
mlb4b@virginia.edu, ejb4n@virginia.edu

***Abstract*— Task analytic models are extremely useful for human factors and systems engineers. Unfortunately, there is no standard language for describing task models. We present an xml-based task analytic modeling language. The language incorporates features from Operator Function Model and extends them with additional task sequencing options and conditional constraints. This language's use is illustrated via a radio alarm clock example. In addition, parsing, visualization, and development tools are discussed.**

***Keywords*— task analysis, XML, operator function model, human behavior modeling**

I. INTRODUCTION

Task analytic models are used to model human task behavior as sequences of activities with respect to the fulfillment of goals. These models have been used in the evaluation of single and multiple operator systems for a variety of purposes including intent inferencing [1], usability evaluation [2], intelligent tutoring [3], timing analysis of human tasks [4], hazard monitoring [5], formal verification of human-interactive systems [6][7], and controlling agents in simulations [8].

Task analytic models have a number of similar features. They can be heterarchical in that they can define multiple independent goals and the tasks to accomplish them. Task analytic models are hierarchical as they decompose high level activities, tasks, or goals into lower level ones. The lowest level in the hierarchy includes atomic level actions such as physical actions (e.g., pressing keys, moving a computer mouse, pressing buttons) and perceptual and cognitive operations. Model specifications may include dependencies within decomposition components such as task ordering. Other types of dependencies are associated with external or internal states or conditions that can determine under what circumstances portions of the model are relevant. Model components can often be reusable as well; where they can be referenced in other models, or in other parts of the same model. Models can also have graphical notations that can be used to represent modeled behavior visually.

Specific task modeling techniques may implement these features differently. For example, Keystroke-Level Modeling Goals, Operators, Methods, and Selection rules (KLM-GOMS) decomposes tasks into sequences of keystrokes [4] without supporting additional hierarchical structures. The Operator Function Model (OFM) [1], ConcurTaskTrees (CTTs) [9], Procedure Representation Language (PRL) [10], and other variants of GOMS [4] allow for multiple levels of decomposition: where any activity can decompose into lower level activities [10]. However, while OFMs, CTTs, and GOMS models ultimately require that activities decompose into atomic actions, PRL does not.

Modeling techniques also differ in how they represent atomic actions. In some version of GOMS (such as KLM-GOMS), all atomic actions must be observable human actions like keystrokes [4]. However, OFMs, CTTs, and Critical Path Method – GOMS (CPM-GOMS) have been used to model non-observable cognitive and perceptual acts such as observing phenomena on an interface or monitoring a particular display [1][4][8][12].

Different modeling techniques vary in how the cardinality and order of activities and actions are specified. OFM supports a number of different cardinality and ordering options (a superset of those supported by PRL) for the sequential execution of activities or actions in each hierarchical decomposition [1][11]: one or more acts (an activity or action) can be executed in any order, exactly one act can be executed, all acts must be executed in any order, and all acts must be executed in a specific order. CTTs support all of these but also support the optional execution of acts (where zero or more acts could be executed), and the synchronous execution of acts (where the acts must be performed at the same time, like pressing two keyboard keys at once) [9]. CPM-GOMS supports the parallel execution of activities in which multiple activities (and their sub-acts) can be executed in parallel in order to encompass human multitasking behavior [4].

The modeling techniques also differ in how strategic knowledge is handled (when goals and activities can or should be considered). Selection rules in GOMS models describe when a particular activity is used [4]. CTTs do not directly implement constraints on an activity's execution, but do contain operators which allow one act to enable another act [9]: behavior that would be handled by conditions in an OFM. OFM supports a number of conditions which can be used to constrain how an activity executes [1][11]:

- *Initiators* specify under what condition an activity should start executing;
- *Conditions to initiate* specify what must be true for an activity to start executing;

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

- *Terminators* specify under what conditions an activity must stop executing; and
- *Conditions to complete* specify what will be true after an activity has completed.

PRL supports a number of conditions that both encompass and extend the conditions utilized by OFM [10]:

- *Start* and *preconditions* specify what must be true for an activity to start executing;
- *Repeat-until conditions* specify what must be true for an activity to cease repeating execution;
- *Invariant conditions* specify what must be true throughout the execution of the activity; and
- *End* and *post-conditions* specify what is expected to be true after the execution of an activity.

Different modeling techniques are often associated with a specific implementation which can restrict their use. For example, OFMs are associated with OFMspert [1][11] and CTTs are associated with CTTE [12]. While PRL is implemented as a platform-independent XML language, it is designed expressly for specifying procedures for spaceflight operations [10]. While GOMS models are represented using an internal notation, a number of different implementations of GOMS exist [4]. Thus, GOMS models are often wedded to specific analysis tools such as CogTool [13].

The graphical representations used for the different models also vary. While PRL does not support a graphical notation, others do. CPM-GOMS models are modeled as pert charts, with operators (or tasks) represented by boxes and links between them indicating dependency [4]. CTT models represent activities in a hierarchy of symbols, where activities can be decomposed into lower-level activities and actions in a tree structure [9][12]. Ordering and inter-activity conditions are represented as operators between peer activities in a decomposition [1][11]. OFMs also represent their task models hierarchically, where atomic actions are presented as rectangles and activities as rounded rectangles. A decomposition is displayed as a large rectangle below the decomposing activity which contains all of the sub-activities or actions. A line connecting the decomposing activity to the decomposition rectangle is annotated with an operator that describes the ordering of the contained activities or actions. Conditions on activities are represented as shapes (a separate shape is associated with each type of condition) attached to it via a line annotated with the condition's logic or, depending on the context of its use, a plain language description.

This paper describes a task modeling language (the Enhanced Operator Function Model (EOFM) language) designed to support both the common and discrepant features of the existing modeling systems. This paper specifies the requirements for a generic modeling system for human task behavior, describes the EOFM language which was designed to meet these requirements, illustrates the application of the EOFM language via an alarm clock programming example, and discusses tools that facilitate the language's use.

II. Language Requirements

Because a generic language for modeling human task behavior would attempt to encompass all of the behaviors of existing task modeling paradigms, it would support all of the requirements described below.

A. Functional Requirments

1) Language models should be capable of representing observable atomic human actions: All of the discussed task models encompass sequences of atomic, observable human actions in fulfillment of a purpose or goal. Thus a generic task modeling language must allow atomic, observable human actions to be modeled.

2) Language models should be capable of representing internal human behaviors: Some task modeling paradigms (including OFMs, CTTs, and CPM-GOMS) allow perceptual or cognitive acts to be modeled, such as noticing an alert, or remembering a quantity displayed on a human-system interface. Thus, a generic task modeling language must provide a means of modeling cognitive and perceptual human acts.

3) Language models should support hierarchical decomposition of goals into activities, multiple levels of activities, and activities into actions: All of the discussed task modeling paradigms are composed as activities which decompose into lower level activities and, potentially, into atomic actions. Thus, a generic task modeling language must provide a means of modeling activities such that they can be decomposed into both lower level activities and atomic actions.

4) Language models should support specification of sub-activity and action execution cardinality: The discussed task models support the ability to describe how many activities within a given decomposition can execute: zero or more, one or more, or all of them. Thus, a generic task modeling language must support all of these cardinalities.

5) Language models should support human behavior temporal orderings: The discussed task models support a number of temporal orderings for describing when activities or acts can be performed: executed in a particular sequential order, performed one at a time in any order, performed synchronously (at the same time), or performed in parallel (any interleaving of activities or actions with potential overlap). A generic task modeling language should support the all of these orderings.

6) The language should be capable of describing constraints/conditions on task execution: Because the execution of a given activity within a task model may be dependent on system, environmental, cognitive, or perceptual conditions, many task modeling paradigms (GOMS, OFM, and PRL) specify what conditions must be true at different phases of a given activity's execution. All of the following conditions are supported by the discussed task models:

- Conditions that force the start of an activity
- Conditions that must be true before starting an activity
- Conditions that must be true throughout the execution of an activity
- Conditions that indicate when to repeat an activity

- Conditions associated with the successful completion of an activity
- Conditions forcing the termination of an activity

Thus, a generic task modeling language should support the ability to specify the logic for all of these conditions.

7) The language should be capable of modeling multiple operators: Some systems have only one human operator while others have multiple operators who are working independently or in cooperation with each other. Thus, a generic task modeling language must provide a means of modeling the task behavior of one or more human operators.

8) Language models should be capable of linking to information from external sources: Task models are often used concurrently with other models or system components, where the state of these other entities may influence what tasks can or should be performed by the operator described by the task model. As such, a generic task modeling language must provide means of communicating information from external sources to task models.

9) Model components should be reusable: Existing task modeling languages support the reuse of activities between and within task models. Such a feature allows for the compact representation and rapid development of models. Thus, a generic task modeling language should support the reuse of activities and actions.

10) Language models should support a graphical notation: Some task modeling systems such as GOMS, CTT, and OFM have graphical notation for visualizing their model's task hierarchical structure, activity orders, cardinalities, and constraints. Thus, a generic task modeling language should support a graphical representation capable of depicting these features.

B. Technological Requirements

To avoid the segmentation associated with the tool specific implementations and non-standard notations, a generic task modeling language should also have the following technological requirements:

- The language should be platform and analysis-environment independent.
- The language should support interpretation or parsing capabilities that will facilitate the incorporation of its models into different analysis infrastructures.

III. Language Description

In addition to the features already discussed, OFM supports a visual and object-oriented means of representing task models. It has state and derived variables that can be used to specify model behavior including its handling of input and output. Thus, it was used as a starting point for the language description and the resulting extension is called Enhanced OFM (EOFM).

The EOFM language utilizes the extensible markup language (XML) [14], a highly supported specification for arranging data in hierarchical structures of nodes which can contain formatted data and valued attributes. Because XML is an international standard, its adoption supports third party applications' ability to parse models represented as EOFMs. The structure of an EOFM XML document was specified using RELAX NG (REgular LAnguage for XML Next Generation) [15], an international standard XML schema language (itself based on XML). A visualization of this specification appears in Figure 1 and is discussed in the subsequent sections.

A. The Root Node

All XML documents must contain a single root node whose attributes and sub-nodes define the document. For the EOFM specification, the root node is called *eofms*. At the next level of the hierarchy, it has azero or more *constant* nodes, zero or more *type* nodes, and one or more *operator* nodes.

B. Types and Constants

Types and constants each serve to help connect EOFMs to the external system and to express model concepts using intuitive descriptors.

Because human operators often interact with interfaces whose component states do not readily translate to variable values associated with the types inherent to most development environments (Boolean variables, integers, floating point values, strings, etc.) the use of custom types can help make models much more intuitive. For example, a special enumerated type could be created to represent the state of a switch with more than two positions, where each position would be modeled with a specific name representing the value selected by the switch. Types can also be used for representing restricted ranges of values. In the EOFM language, such a construct is defined by a *type* node, which itself is composed of a unique *name* attribute (giving it a unique identifier from which it can be referenced by the rest of the document) and a string of data representing the type construction (in the current implementation there are no restrictions on the format of the type construction).

Constants (represented by *constant* nodes) are variables meant to store fixed values. These can be used to communicate bounds on ranges and other unchanging values for types as well conditional logics and variable definitions (both discussed later). In the EOFM language, a constant is defined by a unique *name* attribute, either a *type* attribute (the *name* attribute of a type node) or *othertype* attribute (the name or type construction of a *type* not defined in the xml document which can be used for basic types such as integers, Boolean variables, etc.), and a data string representing the value of the constant.

C. Operators

The *operator* nodes are used to represent the different human operators of the systems (one *operator* node for each human operator being modeled). The task behavior of the *operator* node is defined by and within its sub-nodes. The *operator* node is defined by a unique *name* attribute, zero or more input variables (*ivariable* or *ivariablelink* nodes), one or more human actions (*humanaction* nodes), zero or more local variables (*lvariable* nodes), and one or more enhanced operator function models (*eofm* nodes).

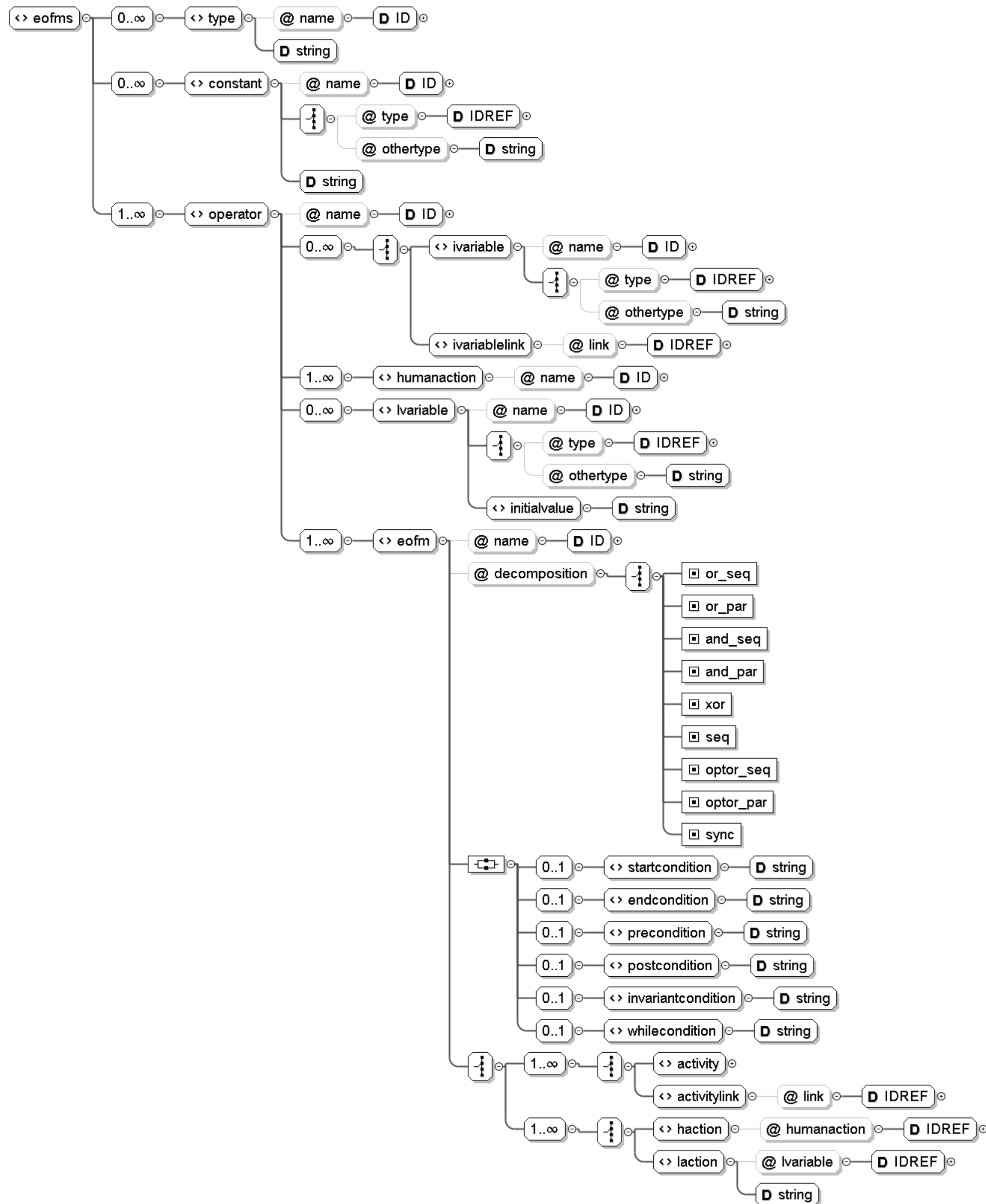

Figure 1. The RELAX NG visualization of the XML-based EOFM language.

D. *Input Variables and Local Variables*

Input variables (*ivariable* or *ivariablelink* nodes) and local variables (*lvariable* nodes) can be used to express the constraints/conditions which control when activities can execute.

Input variables are specifically meant to convey information about external sources (human-system interfaces, environmental data, mission directives, etc.) to the task behavior model. An *ivariable* node is used to define an input variable. It is composed of a unique *name* attribute and either a *type* or *othertype* attribute (defined the same as the identically named attributes of the *constant* node). Because multiple operators can be aware of the same external information (like hearing the same alarm issued by an automated system) the *ivariablelink* node allows an *operator* node to access input variables defined in a different *operator* node. This is facilitated by a *link* attribute which indicates the name of the *ivariable* node being linked.

Local variables are the means by which internal human acts are expressed: where a perceptual or cognitive act is represented by a value being assigned to a local variable. Local variables are represented by *lvariable* nodes, themselves defined with the same attributes as an *ivariable* or *constant* node, but containing an additional sub-node, *initialvalue*, which represents a data string for the variable's default initial value.

E. Human Actions

A human action (a *humanaction* node) describes a single, observable, atomic act that a human operator can perform such as pressing a button. All potential human actions are described in this part of the model so that they can be referenced later in the task behavior description. This node is defined by a unique *name* attribute, presumably describing the action it represents.

F. Task Models

Each *eofm* node represents a single task behavior model at the goal level of the task model hierarchy. Each encompasses execution constraints as well as the hierarchical and temporal relationships between activities and human actions. Structurally the *eofm* node is represented by a unique *name* attribute, a *decomposition* operator attribute, a set of optional conditions, and either one or more activities (*activity* or *activitylink* nodes) or one or more atomic actions (*haction* or *laction* nodes).

The *decomposition* attribute specifies a decomposition operator which controls the temporal execution order of a given *eofm*'s sub-*activity* and *action* nodes (henceforth referred to as sub-acts). In order to support the required combinatorial temporal orderings, the EOFM language implements all of the following decomposition operators:

- *or_seq* – One or more of the sub-acts must execute for the parent *eofm* or *activity* to finish and each sub-act must be executed one at a time.
- *or_par* - One or more of the sub-acts must execute and each sub-act can be executed concurrently.
- *and_seq* – All of the sub-acts must execute and each sub-act must be executed one at a time.
- *and_par* - All of the sub-acts must execute and each sub-act can be executed concurrently.
- *xor* – Exactly one sub-act must execute.
- *seq* – All of the sub-acts must execute, each sub-act must execute one at a time, and each must execute in the order it appears in the markup.
- *optor_seq* – Zero or more of the sub-acts must execute and each sub-act must be executed one at a time.
- *optor_par* – One or more of the sub-acts must execute and each sub-act can be executed concurrently.
- *sync* – All sub-acts must be executed synchronously (at the same time). This is different from the _seq suffixed decompositions as all decomposed actions must be executed at the same time rather than arbitrarily interleaved with each other.

The set of optional conditions provides the means of constraining task model execution. The EOFM language supports all of the following conditions (each represented as a string encompass a Boolean expression written in terms of the defined variables):

- *startcondition* – when true, execution must start
- *endcondition* – when true, execution must terminate
- *precondition* – must be true for execution to start
- *postcondition* – will be true when execution has stopped
- *invariantcondition* – must be true during execution
- *whilecondition* – when true, execution can repeat

The *activity* nodes represent lower-level or sub-activities. They are defined identically to *eofm* nodes, thus allowing for the hierarchical composition of task structures. Activity links (*activitylink* nodes) allow for reuse of model structures by linking to existing *activity* or *eofm* nodes via a *link* attribute which lists the *name* attribute of an *activity* node.

The lowest level of the task model hierarchy is represented by observable, atomic human actions (represented by *haction* nodes) and internal human actions (represented by *laction* nodes). The *haction* nodes are defined by an attribute called *humanaction* which references the *name* attribute of a *humanaction* node. The *laction* node defines a valuation of a local variable. This is defined structurally by an attribute called *lvariable* which references the *name* attribute of an *lvariable* node and a data string representing the value to be assigned to the variable.

G. Graphical Representation

The graphical representation of models written in the EOFM language is based on that used to depict OFMs [1][11]. Actions are represented as rectangles and activities are represented as rounded rectangles. Higher level activity's decomposed activities or actions are depicted in a large rectangle shown below it, connected to the decomposing activity via a line annotated with its decomposition operator. Conditions on activities are represented as shapes and arrows (annotated with the condition logic) that point to the activity that they constrain. The form, position, and color of the shape are determined by the type of condition:

- *startcondition* – A green ball connected to the right side of the activity;
- *endcondition* – A red octagon connected to the left side of the activity;
- *precondition* – A yellow, downward-pointing triangle connected to the right side of the activity;
- *postcondition* – A pink, upward-pointing triangle connected to the left side of the activity;
- *invariantcondition* – A blue square connected to the right side of the activity; and
- *whilecondition* – An arrow attached to the top of the activity with both ends of the arrow connected to the activity.

IV. Example

To illustrate some model concepts, we implemented EOFMs for programming a radio alarm clock (Figure 2). This clock displays the time via a digital readout on its face. The AM and PM designations are indicated by LEDs on the right side of the face. The clock has two alarm functions, a buzzer

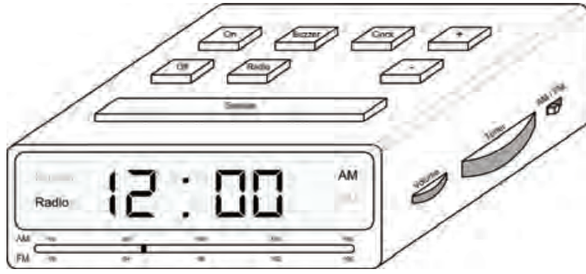

Figure 2. Clock radio used for illustrating modeling concepts

and a radio which are programmed separately. The radio output is separate from the alarm, so both can sound simultaneously.

The tuning of the radio station is controlled by the AM/FM switch and frequency dial on the side of the clock. The selected radio frequency is indicated below the face of the clock. Radio and buzzer volume is controlled by a volume dial on the side of the clock. The radio is turned on and off using the on and off buttons respectively.

An alarm is programmed by pressing and holding either the buzzer or radio buttons (an act which activates the respective LEDs on the clock face). While the button is held, the time for the respective alarm is displayed. The plus (+) or minus (-) buttons increment or decrement the displayed time. Releasing the buzzer or radio buttons sets the respective alarm. An alarm is deactivated (prevented from going off) by pressing the alarm button (either radio or buzzer), pressing the off button, and releasing the alarm button. Similarly, the clock time is set by holding the clock button down and pressing the plus and minus buttons to change the time.

When an alarm goes off, the human operator can either press the snooze button to delay the alarm for 7 minutes or press the off button to turn off the alarm.

A complete listing of the radio alarm clock EOFM can be found at http://cog.sys.virginia.edu/formalmethods/. What follows is a description of some of the implementation details of this model.

This EOFM model had five enumerated types (there were no constants) defining specific values relevant to the information displayed by the clock and the state of its human interface:

- *tOffOn* for the state of clock LEDs: *Off* or *On*;
- *tAMFM* indicating the state of the AM/FM switch: either *AMfreq* or *FMfreq*;
- *tDisplayMode* indicating the display mode on the clock's face: *TimeDisplay* for displaying the time, *SetTime* for setting the clock's time, *SetRadioAlarm* for setting the radio alarm time, and *SetBuzzerAlarm* for setting the buzzer alarm time; and
- *tAlarmState* for indicating the state of an alarm: *Silent*, *Radio*, or *Buzzer*.

To illustrate how these were constructed, the code definition of *tAlarmState* follows:

```
<type name="tAlarmState">{Silent, Radio, Buzzer}</type>
```

The clock has a single human operator (and hence one *operator* node). This operator has no associated local variables but it does have a number of input variables, representing the state of the information displayed on the clock (top of TABLE I) and the desired/mission-directed settings for the clock (bottom of TABLE I).

A number of *humanaction* nodes specify each of the possible atomic programming acts: pressing the on, off, snooze, plus (+), and minus (-) buttons (*hPressOn*, *hPressOff*, *hPressSnooze*, *hPressPlus*, and *hPressMinus* respectively); holding down and releasing the buzzer, radio, and clock buttons (*hHoldBuzzer*, *hReleaseBuzzer*, *hHoldRadio*, *hReleaseRadio*, *hHoldClock*, *hReleaseClock*); flipping the AM/FM switch (*hFlipAMFMSwitch*); and turning the tuning and volume knobs up and down (*hTurnTuningKnobUp*, *hTurnTuningKnobDown*, *hTurnVolumeKnobUp*, *hTurnVolumeKnobDown*).

Twelve *eofm* nodes are used to describe twelve separate goal level activities: setting the clock time, setting the radio alarm time, setting the buzzer alarm time, turning the radio on, setting the AM/FM switch, setting the radio frequency, responding to an alarm (turning the alarm off or activating snooze), setting the volume, activating the buzzer alarm, activating the radio alarm, deactivating the buzzer alarm, and deactivating the radio alarm.

The *eofm* node for setting the time of the buzzer alarm (*aSetBuzzerAlarm*) is shown in Figure 3 along with its visualization in the EOFM graphical notation. This *eofm* has a precondition specifying that it cannot execute until the display mode of the clock (*iDisplayMode*) is *TimeDisplay*. The buzzer alarm can be set by sequentially performing the three subactivities: selecting the buzzer alarm mode (*aSelectBuzzerAlarmMode*), changing the buzzer alarm time (*aChangeBuzzerAlarmTime*), and exiting the mode for setting the buzzer alarm (*aExitBuzzerAlarmMode*). The activity of selecting the buzzer alarm mode has a *precondition* that the display is in time dis-

TABLE I. INPUT VARIABLES (*IVARIABLE* NODES) FOR THE CLOCK PROGRAMMING EXAMPLE

Input Variable Name	Type	Description
iTime	time	The time displayed on the clock
iRadioAlarmSet	*tOffOn*	The state of the Radio LED
iBuzzerAlarmSet	*tOffOn*	The state of the Buzzer LED
iAMFrequency	real	The selected AM Frequency
iFMFrequency	real	The selected FM Frequency
iVolume	real	The volume
iAtMinFreqPosition	Boolean	True if at the bottom of the radio dial
iAtMaxFreqPosition	Boolean	True if at the top of the radio dial
iAtMaxVolumePosition	Boolean	True if at the minimum volume
iAtMinVolumePosition	Boolean	True if at the maximum volume
iDisplayMode	*tDisplayMode*	The mode of display for the clock
iFrequencyMode	*tAMFM*	State of the AM/FM switch
iCurrentTime	time	The actual time
iRadioAlarmTime	time	The desired time for the radio alarm
iBuzzerAlarmTime	time	The desired time for the buzzer alarm
iDesiredFrequency	real	The desired radio frequency
iDesiredFrequencyMode	*tAMFM*	The desired radio frequency mode
iDesiredVolume	real	The desired volume

A.

```
<eofm name="aSetBuzzerAlarm" decomposition="seq">
  <precondition>iDisplayMode = TimeDisplay</precondition>
  <activity name="aSelectBuzzerAlarmMode" decomposition="seq">
    <precondition>iDisplayMode = TimeDisplay</precondition>
    <postcondition>iDisplayMode = SetBuzzerAlarm</postcondition>
    <haction humanaction="hHoldBuzzer"/>
  </activity>
  <activity name="aChangeBuzzerAlarmTime" decomposition="xor">
    <whilecondition>iTime /= iBuzzerAlarmTime </whilecondition>
    <haction humanaction="hPressPlus"/>
    <haction humanaction="hPressMinus"/>
  </activity>
  <activity name="aExitSetBuzzerAlarmMode" decomposition="seq">
    <precondition>iTime = iBuzzerAlarmTime and iDisplayMode = SetBuzzer</precondition>
    <postcondition>iDisplayMode = TimeDisplay</postcondition>
    <haction humanaction="hReleaseBuzzer"/>
  </activity>
</eofm>
```

B.

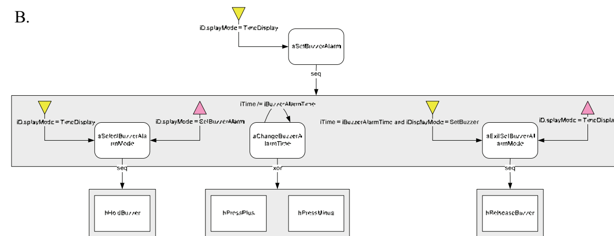

Figure 3. A. EOFM code of the *eofm* node for setting the buzzer alarm. B. Visual representation of the aSetBuzzerAlarm *eofm*.

play mode (*iDisplayMode* = *TimeDisplay*) and a *postcondition* indicating that the activity has completed when the clock is in the display mode for setting the buzzer time (*iDisplayMode* = *SetBuzzerAlarm*). The activity ultimately decomposes into the human action of holding down the buzzer button (*hHoldBuzzer*).

The activity for changing the buzzer alarm time is performed repeatedly as long as the *whilecondition* (that the displayed time is not equal to the desired or mission directed buzzer time: $iTime \neq iBuzzerAlarmTime$) is satisfied. The activity is performed by the human action of pressing either the plus button (*hPressPlus*) or the minus button (*hPressMinus*).

The activity for exiting the display mode has the *precondition* that the display be in the mode for setting the buzzer alarm time and that the displayed time be equal the desired buzzer alarm time (*iTime* = *iBuzzerAlarmTime* and *iDisplayMode* = *SetBuzzer*) and a *postcondition* indicating that the action has completed when the time display mode is shown (*iDisplayMode* = *TimeDisplay*). The activity is completed by performing the action of releasing the buzzer button (*hReleaseBuzzer*).

The EOFM language and this example are currently being disseminated via http://cog.sys.virginia.edu/formalmethods/. Here one can download the EOFM RELAX NG schema, the clock radio example, and the full visualization of the clock radio example's EOFMs.

V. Discussion

The EOFM language introduced in this paper meets all of the requirements laid out in the introduction:

- The language allows for the direct modeling of atomic human actions via references to *humanaction* nodes.
- The language allows for the modeling of internal human actions through valuations of local variables (*lvariable* nodes).
- The language allows models to be constructed hierarchically using *eofm* and *activity* nodes which decompose into other *activity* nodes and, ultimately, atomic actions (*haction* and *laction* nodes).
- The language allows for the specified cardinalities and temporal orderings (those supported by OFM, CTT, GOMS, and PRL) through the use of decomposition operators.
- The language allows for the conditional executions of activities given in the specification, supporting all of the conditions found in OFM, CTT, GOMS, and PRL.
- The language allows for the modeling of multiple operators via multiple *operator* nodes.
- The language is capable of allowing models to receive information from external sources through the use of custom types, constants, and input variables.
- The language allows activities to be reused between and within task models via *activitylink* nodes.
- The language is specified in RELAX NG and implemented in XML which fulfill the technological requirements: both are platform independent and XML documents are easily parsable with existing code libraries.
- The language supports a graphical notation for visually describing model activity and action hierarchies with the model-specified decomposition operators and conditions.

Since the EOFM language met its technological requirements through the use of RELAX NG and XML, a number of tools associated with these technologies can be used to facilitate the development and use of the EOFM language.

A. Tool Support

1) Integration with Existing Technologies: The parsing capabilities offered by existing XML libraries allow the EOFM language to be integrated into a number of existing tools and frameworks. For example, we have developed a Java-based parser that translates EOFM models into the language of the Symbolic Analysis Laboratory, allowing human task behavior to be verified as part of a formal system model [7]. We have also developed a Visio document which uses Microsoft's XML parsing libraries to render visualization of the EOFM models as Visio drawings (for example, Figure 3B).

2) Automatic Generation of Software: A number of tools exist for expediting the production of parsing software for XML based languages. For example, Relaxer and RelaxNGCC can be used to automatically generate java code to parse RELAX NG specified languages (see http://relaxng.org/). Thus others can create their own EOFM code parsers.

3) Model Validation: Many parsing libraries for XML have the capability to validate that an opened XML document

matches a given RELAX NG schema. Thus EOFM modelers should be able to check that their developed models are properly conforming to the EOFM language structure.

4) Integrated Development Environments: A variety of XML integrated development environments (IDEs) such as the oXygen xml editor (see http://www.oxygenxml.com/) and nXML for Emacs (see http://www.thaiopensource.com/nxml-mode/) can be used to dynamically check that an XML document is conforming to a RELAX NG schema while it is being written. Such programs can also use schemas to suggest nodes, node attributes, and data values dynamically via code-completion functionality. Thus, these environments can be (and have been) used as effective IDEs for EOFM models.

B. Future Work

There are some ways the EOFM language could be improved. Firstly, the current implementation does not specify a particular syntax for type constructions, initial values, and condition formulas. RELAX NG offers a variety of ways in which this could be done (additional XML structure and/or regular expressions specifying data formatting). Future work should investigate this potential feature.

Some forms of the operator function model allow for conditions to be established between two activities in any given decomposition, allowing for execution to shift between activities much like a finite state machine [1]. While the current implementation of EOFM can use its supported conditions to produce a model that exhibits comparable behavior (an *endcondition* on the source activity and a *startcondition* on the destination activity with the same condition logic are equivalent to a transition from one *activity* to the other), it may be a less intuitive representation. Future work should investigate whether inter-activity, transitional conditions are a useful feature.

Work by Lee and Sanquist has attempted to extend the OFM by adding cognitive operations [16]. Future work should more explicitly attempt to incorporate these capabilities into the EOFM language.

Finally, the EOFM language is intended to be easy to use by the Human Factors and Systems Engineering communities. As such, it may be necessary to conduct studies to determine what application areas and analyses the language best supports and thus find ways of improving the capabilities of the language. Further, it may be useful to conduct usability analyses to assess how intuitive and easy to use the language is and investigate potential improvements that would facilitate such goals.

C. Conclusions

The EOFM language offers a complete, flexible, generic, platform-independent, means of creating human task models that can be parsed and incorporated into other infrastructures. Through the use of existing XML technologies, the EOFM language can be used with sophisticated pre-existing development resources which facilitate its use.

Acknowledgement

The project described was supported in part by Grant Number T15LM009462 from the National Library of Medicine and Research Grant Agreement UVA-03-01, sub-award 6073-VA from the National Institute of Aerospace (NIA). The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIA, NASA, the National Library of Medicine, or the National Institutes of Health.

References

[1] Thurman, D. A., Chappell, A. R., and Mitchell, C. M., “An enhanced architecture for OFMspert: a domain-independent system for intent inferencing,” *IEEE International Conference on Systems, Man, and Cybernetics*, vol.1, no., pp. 955-960 vol.1, 11-14 Oct 1998.

[2] Lecerof, A. and Paternò, F., “Automatic support for usability evaluation,” *IEEE Transactions on Software Engineering*, vol.24, no.10, pp.863-888, Oct 1998.

[3] Chu, R. W., Mitchell, C. M., and Jones, P. M., “Using the operator function model and OFMspert as the basis for an intelligent tutoring system: towards a tutor/aid paradigm for operators of supervisory control systems,” *IEEE Transactions on Systems, Man and Cybernetics,* vol.25, no.7, pp.1054-1075, Jul 1995.

[4] John, B. E. and Kieras, D. E., “Using GOMS for user interface design and evaluation: Which technique?” *ACM Transactions on Computer-Human Interaction*, vol. 3, pp. 287-319, 1996.

[5] Bass, E. J., Ernst-Fortin, S. T., Small, R. L., and Hogans, J., Jr., "Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-base development," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* , vol.34, no.4, pp. 441-449, July 2004.

[6] Fields, R. E., “Analysis of erroneous actions in the design of critical systems,” D. Phil Thesis, Technical Report YCST 20001/09, University of York, Department of Computer Science, 2001.

[7] Bolton, M. L. and Bass, E. J. “A Method for the Formal Verification of Human-interactive Systems,” *HFES 53rd Annual Meeting*, San Antonio, Texas, October 19-23, 2009.

[8] Göknur, S., Bolton, M. L., and Bass, E.J. “Adding a motor control component to the Operator Function Model Expert System to investigate air traffic management concepts using simulation,” *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp.886-892, October 10-13 2004.

[9] Paternò, F., *Model-based Design and Evaluation of Interactive Applications*. Springer Verlag, 1999.

[10] Kortenkamp, D., Dalal, K. M., Bonasso, R. P., Schreckenghost, D., Verma, V. and Wang, L. “A Procedure Representation Language for Human Spaceflight Operations”, *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space,* 2008.

[11] Mitchell, C. M., Thurman, D. A., Brann, D. M., and Chappell, A. R., “OFMspert I: operations automation,” *IEEE International Conference on Systems, Man, and Cybernetics*, vol.2, no., pp.1093-1098 vol.2, 2000.

[12] Mori, G., Paternò, F., and Santoro, C., “CTTE: support for developing and analyzing task models for interactive system design,” *IEEE Transactions on Software Engineering*, vol.28, no.8, pp. 797-813, Aug 2002.

[13] John, B. E. and Salvucci, D. D., “Multipurpose prototypes for assessing user interfaces in pervasive computing systems,” *Pervasive Computing, IEEE*, vol.4, no.4, pp. 27-34, Oct.-Dec. 2005

[14] “Extensible Markup Language (XML) 1.0 (5th Edition) W3C Recommendation,” Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. Eds., November 2008, Available: http://www.w3.org/TR/2008/REC-xml-20081126/

[15] Clark, J. and Makoto, M., *RELAX NG Specification*, OASIS, 2001, Available: http://relaxng.org/spec-20011203.html

[16] Lee, J. D. and Sanquist, T. F., "Augmenting the operator function model with cognitive operations: assessing the cognitive demands of technological innovation in ship navigation," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* , vol.30, no.3, pp.273-285, May 2000