Published in IEEE Transactions on Human-Machine Systems, 46(2):303-316. DOI: 10.1109/THMS.2015.2421511

Formal Verification of a Space System's User Interface with the IVY workbench

José Creissac Campos, Manuel Sousa, Miriam C. Bergue Alves, Michael D. Harrison

***Abstract*—This paper describes the application of the IVY workbench to the formal analysis of a user interface for a safety-critical aerospace system. The operations manual of the system was used as a requirements document and this made it possible to build a reference model of the user interface, focusing on navigation between displays, the information provided by each display and how they are interrelated. Usability related property specification patterns were then used to derive relevant properties for verification. This paper discusses both the modeling strategy and the analytical results found using the IVY workbench. The purpose of the reference model is to provide a standard against which future versions of the interface may be assessed.**

***Index Terms*—Formal verification, IVY workbench, usability.**

I. Introduction

Being able to continue to validate requirements throughout the life of a system is particularly important when a system is safety critical. System updates can include replacement of components and changes to the user interface to reflect new technologies. Formal specification techniques can play a role in this ongoing process by providing a proven and valid reference model and identifying critical properties to be used as criteria for the acceptance of system updates.

A 1994 paper [1] that investigated computer related deaths amongst a range of accident descriptions revealed that 92% of computer related deaths could be related to user interaction failures. Since 1994 user interface designs have become increasingly complex and, even though developers are more sensitized to use failures, user interaction continues to be a major issue. Important problems in the analysis of interactive systems result from the complexity and concealed nature of interaction modes introduced by new technologies. Failures often arise because of mode confusions [2] which are a consequence of poor feedback. While it is possible to explore these mode complexities and confusions using a range of techniques, successful analysis should be systematic and exhaustive. Formal analysis techniques have been developed that provide such analyses. The advantage of these techniques for identifying potential problems is that they are precise, concise and amenable to automated, computer-aided analysis. Their disadvantage is that, although significant progress has made these tools more effective and usable, the scope of analysis is narrow in comparison with traditional usability analysis techniques. The challenge is to develop tools that can be used effectively by developers.

This paper's main goal is to show how a particular formal technique is applied to the user interface of a safety-critical system used within the aerospace domain. A reference model is produced that satisfies a set of user interface properties and can be used as a standard against future versions of the design. The approach is based on model checking techniques using the IVY workbench [3], [4], a model-based tool for the analysis of interactive systems, which has shown its potential to be used in other critical areas [5], [6]. Analysis uses models that are constructed of the user interfaces under analysis. Verification is achieved by proving a set of properties (many based on templates) that express requirements over the specified use of the system.

The system, responsible for the ground control, testing and preparation of a satellite launcher developed by the Brazilian Space Launcher Program, is an evolving legacy system. It has been in use for more than 15 years. Over time, the system's requirements and in particular user interface requirements have remained mostly unchanged. There is a good reason for conservatism. Changes to the user interface are likely to change operating procedures with consequent training costs and potential for user error.

The motivation for the current work is twofold. On the one hand, the existing system is due for replacement and it is understood by the organization that building a thorough understanding of it (including its user interface) will help shape the requirements for the new system, its testing and acceptance. The challenge for the present paper is that requirements should be expressed at an appropriate level, and be capable of formulation as properties, so that evolutions of the design can be shown to satisfy the requirements. A further aim is that the evolution can be tested in relation to the requirements.

On the other hand, the user interface of the system has two features that make it challenging from the perspective of formal modeling. It features a number of workstations with relatively complex displays, each involving a number of overlapping but interdependent panels. Operators are able to switch between these panels and mode confusions can arise as a result. While mode confusions have previously been

Manuscript received —; revised — This work was partly funded by project ref. NORTE-07-0124-FEDER-000062, co-financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese foundation for science and technology (FCT).

J. C, Campos is with Departamento de Informática, Universidade do Minho & HASLab/INESC TEC, Campus de Gualtar, 4710-057 Braga, Portugal. e-mail: jose.campos@di.uminho.pt

M. Sousa is with WeDo Technologies, Braga, Portugal.

M. Alves is with Instituto de Aeronáutica e Espaço (*Aeronautics and Space Institute*), São José dos Campos, Brazil.

M. D. Harrison is with Newcastle University, Newcastle upon Tyne & Queen Mary University of London, Mile End Road, London, UK.

analyzed using the approach to be described in this paper [3], [4], the particular challenge here is how to model the screen and whether the approach used in previous analysis will scale. The other feature is that the system is safety critical and therefore verification and validation is typically audited independently by a third party. The focus of the analysis is on the interaction between operators and system, and for this reason the formal model of the system is based on the User Operation Manual. The manual provides a detailed account of the user interface of the system and therefore a good basis for the reference model.

The developed model, expressed in the MAL (Modal Action Logic) Interactors language, captures a relevant subset of the system's user interface as a hierarchy of interactors. The paper (as space and non disclosure agreement constraints allow) illustrates part of the model that has been developed. A collection of feedback properties are proved of the system.

In summary, the paper's main contribution is an illustration of how a model developed from an operations manual can be used to reason about the intended design of a system's user interface, and of how the manual made it possible to develop such a model of the user interface without detailed technical information about the system's implementation. As will be illustrated, the model enabled reasoning about properties of the system's user interface. While it is relevant to point out that these properties are proved of the system as described in the operations manuals, the paper also shows how it was possible to uncover missing information in the manual itself; that is, the model and the manual can be tested against an implementation of the concrete system. An additional contribution is the illustration of the scalability of the verification approach used. A preliminary description of this work was published as [7]. The current paper extends it by providing a more detailed account of the process, particularly the analysis and discussion of results.

The paper is structured as follows. Section II discusses the research that provides the background to this analysis. Section III describes the space ground legacy system and Section IV describes the formal approach used. Sections V and VI describe the model of the system and its analysis. Section VII discusses the results. Section VIII concludes the paper.

II. Related work

Most existing analytical and inspection-based usability analysis methods do not provide a style of analysis that is systematic and exhaustive enough to provide adequate guarantees of compliance with given requirements for safety critical systems. Formal methods are able to provide such systematic and exhaustive analysis. Model-checking [8], is one such formal technology that involves exploration of the state space defined by the model of the system (its specification). It can use this exploration to verify that the system model satisfies formulas representing relevant properties. These formulas are expressed in a temporal logic and describe properties over the behavior of the system. When verification fails, the model checker points out (if it exists) a behavior of the system (a counter-example) that provides an example of where the property fails[1].

The applicability of formal methods to the modeling and verification of safety-critical systems such as space systems and other safety critical domains has been extensively explored [9], [10], [11], [12], [13]. This applicability of exhaustive and repeatable analysis of user-system interaction is an active topic of research. Areas of applicability include avionics [14], [2], and the medical field [15], [4], [16], [5].

Bolton et al. [17] have identified two broad categories of approach. Those that focus on the analysis of the user interface of the system, and those that focus on the analysis of how the system is (supposed to be) used. Approaches in the first group, of which our work [14], [3], [4] is an example, typically use a model of the user interface under analysis, proving properties of the interface that are relevant to the operation of the system. Examples of properties include usability principles (e.g. [3], [18], [19]), mode confusion properties (e.g. [20], [14], [2], [21]) and user-related safety requirements [22], [23]. To help focus on user relevant issues and behavior, the inclusion of mental models (models of how the user believes that the system will work) or knowledge models (models of what the user knows about how to use the system) have been used to augment the analysis. Examples of the use of mental models include [2] and [24]. Examples of the use of knowledge models include [25] and [6].

Approaches in the second group (focusing on use) work either with task models of how the users are supposed to use the system (their observed behavior) [26], [27], [28], or with cognitive models of the mental process that drive that behavior [29], [30], [31].

Initially case studies used to develop and/or illustrate the approaches tended to be relatively small devices, raising questions as to whether the techniques would scale. The growing maturity of the area has meant that these techniques and tools can be scaled to real problems [32], [5]. An important consideration when performing analysis is to know which properties are relevant to which systems and how to express these properties in the appropriate logic for verification. The relevance of properties is largely domain dependent relying on knowledge of the system as a whole. At the same time a correct understanding of the model, the requirements, and the logic in which properties are expressed, is needed in order to guarantee that the properties being verified encode the intent of the verification process. As illustrated by [33], many situations can be found in the literature where a logical formula used for verification does not correspond to what was intended.

Specification patterns can be used to address the problem of property formulation [33]. These patterns are helpful, not only to support the process of writing correct properties, but also as prompts for potentially relevant properties that should be verified. In the specific case of interactive systems, patterns capturing relevant usability properties (e.g., feedback)

[1]For some properties, providing a counter-example is not possible. Consider, for example, the case of trying to prove that the system can reach some designated state. If that is false, the counter-example would have to be all possible behaviors of the system. In more technical terms, what happens is that for formulas with existential path quantifiers a single behavior of the system is not enough to demonstrate them false.

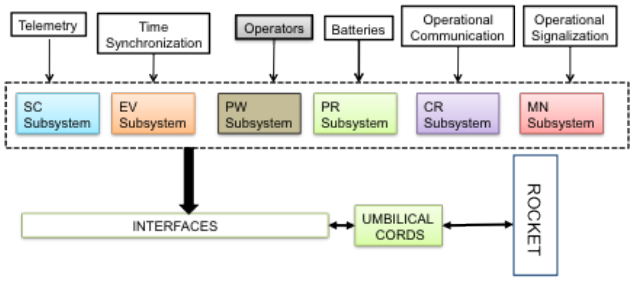

Fig. 1. The TPGS's macro architecture

have been proposed [4]. Alternative approaches include the automatic generation of properties from task models [34].

Patterns are used in this paper to explore a model of a safety critical system's user interface systematically. The aim is to verify that the model exhibits relevant generic properties related to the modeled system's safety as well as its usability. These properties specify requirements that are recognized by regulators as mitigating relevant risks. This model is capable of being used as a benchmark against which the system and its evolution can be assessed. This process should enable identification of relevant aspects of the system that might impact its safety and therefore require further scrutiny.

III. The Testing and Preparation System (TPGS)

The TPGS is a ground based system that has been used to support space mission preparation for more than 15 years as part of the Brazilian Space Launcher Program. It is a legacy safety-critical system containing customized hardware and software components. These components are responsible for ground control, testing and pre-launch preparation of a satellite launch rocket. It consists of six subsystems each providing specific functionality relevant to the rocket's testing and preparation. Each subsystem is operated through a dedicated workstation whose user interfaces comprise both a GUI based interface and a physical panel of buttons and actuators. Figure 1 shows the macro-architecture of the system.

The functional requirements for the TPGS were established separately for each subsystem and are described in a 250-page document. The rockets preparation process for flight includes thoroughly checking specific parts of the rockets electrical network. The operator of each subsystem must follow a comprehensive preparation checklist that performs critical procedures and interactions with the rocket's hardware and software in real-time. The preparation and testing process is triggered by operator interaction with the system via the user interfaces. The GUI of each subsystem must support these procedures and interactions so that the checklist is successfully completed.

The software components of TPGS are often updated as appropriate to accommodate new mission requirements, different rocket configurations, and new operating systems releases. The hardware is also occasionally upgraded as a result of equipment obsolescence. An important goal is that despite these updates operating procedures are changed as little as possible. As main contractors the Space Institute have responsibilities for these activities. A goal of the Space Institute is to provide a set of formally specified and validated critical requirements to serve as a baseline (an oracle) for system acceptance. This goal includes providing a set of user interface requirements for correct mission-safety system operation that do not change as the system evolves through updates and upgrades. It is this role that the reference model is designed to play.

Models of the system's user interfaces derived from the Operator Manual contribute to this goal. This is achieved by establishing a set of human interaction sequences with the system along with the critical properties to be verified within each sequence. The purpose behind this strategy is to run tests on a new or updated system that will enable the clients (e.g. subject matter experts) to use the verified set of user interfaces as a basis for accepting the new operation environment.

The GUIs of two of the subsystems have been modeled so far. The EV (flight events' sequencing) subsystem has been chosen as illustration. This SCADA (supervisory control and data acquisition) system enables the operator to monitor the state of different aspects of the system and relevant flight events during sequence testing and launching. It enables the operator to activate different tests and allows the operator to activate the rocket's security mechanical devices during the automatic sequencing for launching. The EV subsystem's GUI consists of panels, each presenting a different view of the system (examples are telemetry readings and a synoptic view of the system). A row of buttons at the top of the panel allows access to the different panels available in the subsystem. Each panel contains tens of graphical elements representing system variables (examples are a voltage reading and the state of a relay). When a parameter falls outside specified bounds an alert or alarm is raised and the displayed parameter changes color (yellow for alerts, red for alarms) or blinks depending on severity. The button that accesses the panel also changes color or blinks. Two panels are designated to collect and display the alarms and alerts reported in the other panels. More details about the system are provided in Section V.

IV. The Formal Approach

The analysis process used in this case study is supported by the IVY tool. Further details can be found in [14], [3], [4]. The IVY tool supports the MAL interactor language for modeling the systems, the specification of properties, and the presentation of any feedback produced when a property fails. These elements are described in more detail in this section.

A. The MAL language

Models are developed in the MAL Interactor language. The interactor is a structuring object [35] that describes how part of its state is rendered to some presentation medium. More explanation of the notation and its use can be found in Section V. MAL is used because it describes state transitions in a similar form to those specified by graphical notations such as Simulink statecharts. Notations such as these are increasingly acceptable to industry (see [36] for example).

A MAL interactor is defined by:

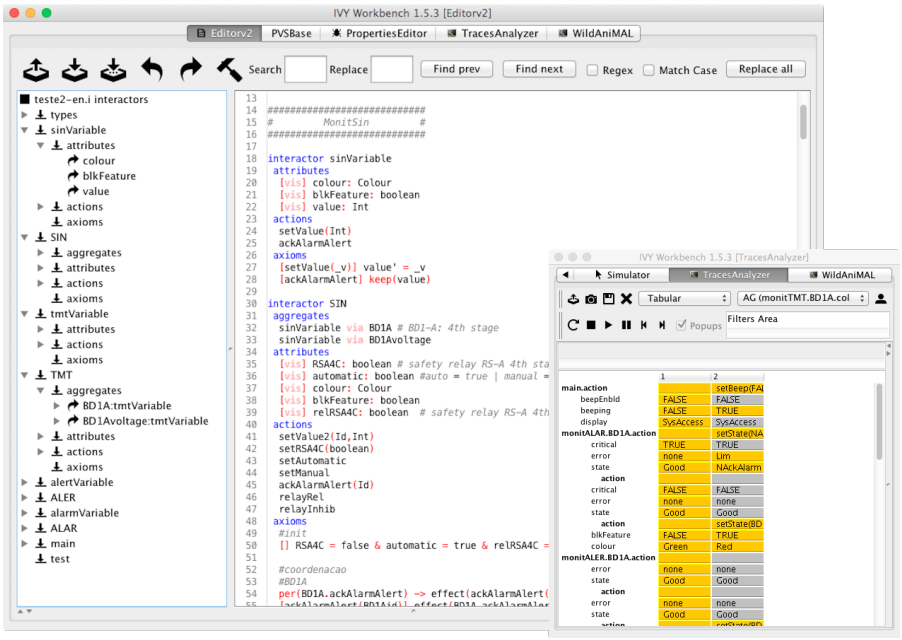

Fig. 2. The IVY tool

- a set of typed attributes that define the interactor's state;
- a mapping of the state to some presentation medium;
- a set of actions that define operations on the state;
- a set of axioms written in MAL that define the semantics of the actions in terms of their effect on the state.

Those parts of the state that are rendered are indicated by decorating the attributes with modality annotations. MAL axioms define how an interactor's state changes in response to interactor actions. Axioms are of three types:

- Propositional axioms – which define axioms that must always be true of the state of the interactor;
- Modal axioms – these axioms define the effect of an action on the state of the interactor;
- Deontic axioms – these axioms define the conditions under which an action is permitted or obligatory.

Models are constructed by composing interactors hierarchically. A model can therefore be seen as a state machine, where the states are defined by the attribute values and the transitions are labelled by the actions that cause changes to the attributes.

B. The CTL logic

Properties for verification are specified in CTL (Computational Tree Logic) [8]. These express assumptions about the expected behavior of the device. The example focuses on a general problem in user interfaces that confusions can occur when modes or other aspects of the system state are not clearly indicated and as a result the behavior of actions is unclear [3].

CTL enables description of quantifiers over paths in the model using either the operator "A" (along all paths) or "E" (along at least one path). Predicates on states in these paths may be expressed using temporal operators "G" (always), "F" (eventually), or "X" (in the next state). Propositional logic is used to express the state properties. By combining quantifiers and predicates it is possible to express properties such as that some state property is true of all states, or that some state property is inevitable. For example:

- $AG(effected(help) \rightarrow display = helpScr)$ means that, for all paths starting at the current state (A), and for all states in those paths (G), the help action $(effected(help)$[2]$)$ implies that attribute $display$ has value $helpScr$
- $EX(display = helpScr)$ means that for some path (E), in the next state in that path (X) the $display$ attribute has value $helpScr$
- $EF(display = helpScr)$ means that for some path (E), in some state in that path (F) the $display$ attribute has value $helpScr$

C. Tool support – The IVY workbench

The IVY tool (see Figure 2) supports the development of models of the interactive device, the formulation of required properties of the device's behavior and their verification through model checking. When verification fails, the counterexamples produced by the verification process act as scenarios for analysis. The tool supports these activities as follows.

1) Building models: Models are developed in the MAL Interactor language. IVY offers a textual editor that supports syntax highlighting, code completion, undo/redo and other usual editing facilities. The editor also presents as a side panel a tree view of the model that allows easy navigation of the model's structure.

2) Expressing and verifying properties: Patterns help express properties. They have been collected from a number of

[2]The *effected* operator is part of MAL, not CTL.

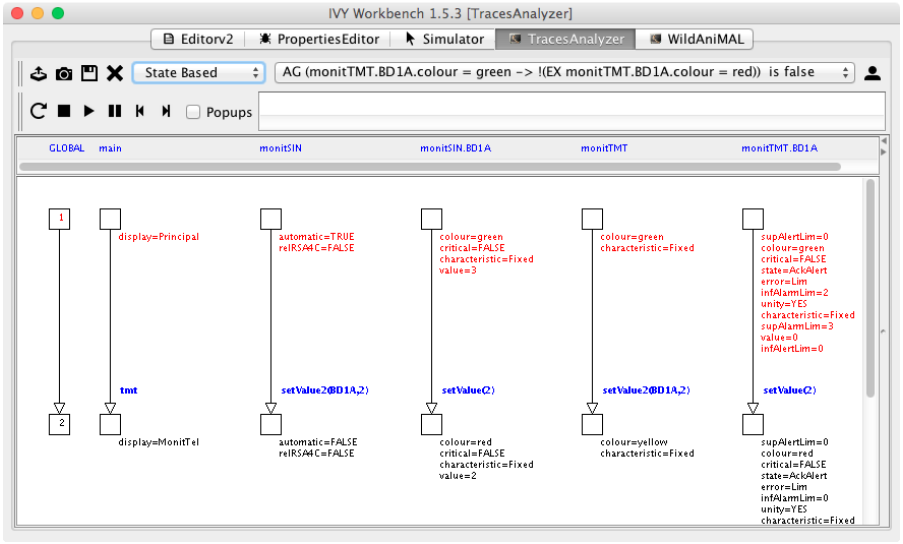

Fig. 3. State-based representation of a trace.

sources [37], [38]. The tool supports the selection and instantiation of patterns, with the actions and attributes of the model, thus facilitating the identification and development of CTL properties that describe requirements. The verification step itself is performed by invoking the NuSMV model checker [39] from the IVY tool. MAL interactor models are translated automatically into equivalent SMV models.

3) Analyzing results: Visual representations of counter-examples where properties fail are offered to provide support for analysis. A tabular representation (see inset in Figure 2) uses columns to represent states and lines for actions and attributes. The yellow/darker background color is used to highlight actions or attributes where values have changed. An alternative state-based representation (see Figure 3) is also offered that represents each interactor by a column that shows the states that the interactor goes through. Actions are indicated by labels on the arcs between two consecutive states.

Properties can be checked using the model checker and refined iteratively to verify the circumstances in which a property is true. If it fails to be true then the trace offers a counter-example. These counter-examples provide three types of information. Property failure can be the result of a deliberate strategy by the analyst, in which the originally property is negated in order to obtain a sequence of actions representing a potentially significant scenario where the required property is satisfied. For example, alternative sequences that involve a confirming action can be used to assess situations in which the confirming action could be used. The failure of a property can also provide valuable feedback to indicate why the modeled system fails the property. This can be useful in redesign, or in deciding that the circumstances in which the property fails are not significant in terms of the correctness and safety of the system. The failure of a property can, finally, indicate that the property is not adequate as its stands to represent the requirement. This kind of failure would result in refinement of the property.

V. Modeling the EV subsystem

A formal model of the EV subsystem offers benefits. It enables a clearer understanding of the system. It also enables exhaustive and systematic analysis of the modeled system. For these benefits to be manifest an appropriate level of abstraction must be taken in the model. The focus of the model in this case is to capture the information and actions that are available through the user interface. A focus on the user interface to the EV subsystem can be achieved without making assumptions about the behavior of the rocket and its supporting systems. The model of the user interface developed for the case study is triggered by these systems randomly. There are of course situations where these other parts of the system are important to an understanding of the user interface as is described, for example, in [40].

Constructing a model from the User Operation Manual that was sufficiently rich to provide meaningful feedback to the Aeronautics and Space Institute (IAE) team offered a significant challenge.

A. Macro-structure of the model

The EV subsystem operation manual describes the EV subsystem as consisting of 23 interdependent panels each featuring output elements exhibiting slightly different behaviors. Interdependency is most evident in the alarms and alerts panels which aggregate information variables that are in an alarm/alert condition from the other panels. Previous analyses using the IVY tool have involved multiple panels (see, for example, [5]) but they have used a flat model structure, employing a single interactor. This single interactor represents the union of all panel information. While this made it simpler to describe coordination between different elements of the panel, it was only capable of dealing with relatively simple display combinations. More structure was sought to achieve a more general approach. The specified model uses the object-oriented features (in particular the aggregation relation) of

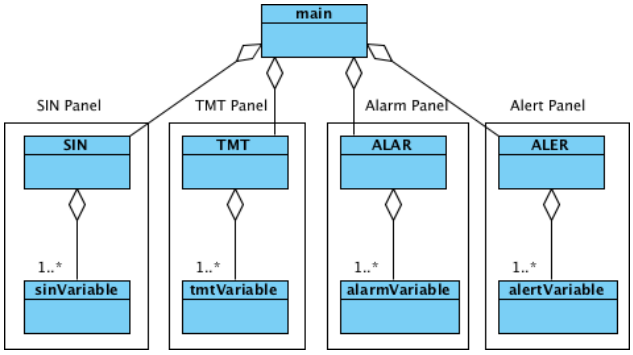

Fig. 4. The models's macro architecture (aggregation is represented, in UML style [41], by a connection with a diamond shape on the aggregator's side).

the MAL interactor language to create a tree-like structure of panels.

Figure 4 shows the architecture of the model describing four of the panels An interactor for each panel is aggregated in the main interactor. Each panel interactor aggregates interactors that represent the state, the display behavior and controls associated with the displayed variables. Many instances of variables are needed to represent the information that a panel displays. This structured approach enables better management of the scalability of the model. However a remaining challenge is expressing coordination between panels.

Although alternative approaches have been explored (cf. [42]), the composition of interactors is currently strictly hierarchical. The *main* interactor coordinates dependencies between all the other interactors. It ensures that changes to the state of a panel variable are reflected throughout the model, including, for example, the *Alarm* and *Alert* panels. These panels provide a summary view of the variables that are in an alarm or alert condition. Furthermore, each interactor representing a panel is responsible for its own internal dependencies. This approach favors an incremental style of modeling, in which detail can be gradually added to each interactor as modeling progresses and/or more information becomes available. This happens both in terms of adding additional panels to the model, and in adding detail to each of the panels. It remains to be seen however how this centralized approach will scale as more variables and panels are added to the model; not only in terms of expressing the coordination, but also in terms of computational resources during verification.

B. Navigation between panels

The user interface of the EV subsystem contains a top row of navigation buttons and a display area where the panels are shown. Which panel is being displayed is captured in the model by an attribute ($display$). The attribute has type $Screens$, representing all panels that compose the subsystem:

types
$Screens = \{MainScr, Events, Proced, Reports, Help, ...\}$
interactor $main$
attributes
$[vis]display : Screens$

The $[vis]$ annotation in the $display$ attribute indicates that the information represented by the attribute is visible; that is, users can see which panel is being displayed.

Navigation buttons are modeled by actions. These actions update the value of the $display$ attribute. For example, the action $sinBtn$ changes the value of $display$ to indicate that the $Synoptic$ panel is visible.

actions
$sinBtn\ tmtBtn\ alertsBtn\ alarmsBtn\ ...$
axioms
$[sinBtn]\ display' = Synoptic$
...

This action is only permitted when the appropriate navigation button appears on the user interface. As described in the manual, the row of navigation buttons is not available on a number of panels, including the login ($SysAccess$) and exit ($EndScr$) panels, and when the system is blocked ($ScreenLock$). For action $sinBtn$, this is specified by the permission axiom:

$$\mathsf{per}\,(sinBtn) \rightarrow\ !(display\ in\ \{SysAccess, SEVaccess, ScreenLock, SelScr, EndScr\})$$

All actions that are used to navigate between panels are specified similarly. To complete the navigation model the initial panel of the system is described. This is done using the initialization axiom:

$$[]display = SysAccess$$

C. Modeling the panels

Each of the panels of the EV subsystem presents information about a specific aspect of the system. Modeling a panel involves defining the the variables represented on the panel along with the additional actions and control logic that relate to them. The manual describes these in detail.

1) Variables: Displayed variables can have behavior; for example, they change color under certain conditions. These variables are captured as interactors which are then aggregated into the panel's own interactor. Simpler variables, where color and display characteristics are not relevant (for example, the state of a relay represented as a boolean value), are included directly as attributes. In most cases the manual describes variables as having a value that can change over time, a color used to display the value, and a display characteristic, which can be either fixed or blinking.

The value of a variable is modified according to underlying system properties (e.g. temperature of some specific device). The model assumes nothing about the processes that the subsystem manages. Therefore, it provides no restriction on the behaviors of variables, which permits an analysis of error behavior.

Variables can be critical or non-critical, depending on which system property they represent. The telemetry monitoring panel (modeled by interactor TMT) is a good example because it includes more complex variables that feature additional characteristics including "working limits," description,

and measurement units. The working limits specify when alerts or alarms can occur. Alerts and alarms must be acknowledged by the operator by selecting the corresponding entry on the alerts or alarms panel. The $tmtVariable$ interactor specifies attributes including value, color and other display characteristics. Changing the variable and its attributes is achieved generically through the $setValue(int)$ action. This action specifies how the panel reacts to changes to the underlying system properties. Axioms defining this action are of a general form, where priming is used to reference the value of an attribute in the state after the action has happened, as follows:

$$\begin{aligned}[setValue(\_v)]\ &(\text{conditions on \_v}) \rightarrow \\ &value' = \_v \\ \&\ &color' = \text{new\_color\_value} \\ \&\ &characteristic' = \text{new\_characteristic\_value}\end{aligned}$$

As an illustration of how the model was generated from the operation manual, consider the following extract (translated from the Portuguese original), which describes when a variable from the telemetry monitoring panel should be displayed in blinking yellow (i.e., $value' = yellow\ \&\ characteristic' = blink$):

> *Blinking yellow: For a critical variable, when the current value of the variable is in non acknowledged alert (value within the alert range), there is no acknowledged alarm in the variable, and the previous criterion [non acknowledged alarm criterion] is not satisfied. If over the same critical variable an acknowledged alarm exists, then Fixed Red prevails. For a non critical variable, when the current value of the variable is a non acknowledged alarm (value within the alarm range)*

Whether a variable is critical or not is represented by the boolean attribute $critical$, and the upper and lower alert and alarm limits by $infAlertLim$/$supAlertLim$ and $infAlarmLim$/$supAlarmLim$, respectively. The condition expressed in the first sentence of the quote above can therefore be expressed as:

$$\begin{aligned}&critical \\ \&\ (\ &(\_v >= infAlarmLim\ \&\ \_v < infAlertLim) \\ &\mid (\_v <= supAlarmLim\ \&\ \_v > supAlertLim)) \\ \&\ &alarmState! = AckAlarm\ \&\ alarmState! = NAckAlarm)\end{aligned}$$

where *AckAlarm* is the acknowledged alarm state and *NAckAlarm* is the non acknowledged alarm state.

In the same way the last sentence of the manual extract can be represented by the expression:

$$!critical\ \&\ ((\_v < infAlarmLim) | (\_v > supAlarmLim))$$

The full axiom representing these two sentences is presented in Figure 5. The type of error and the units associated with the variable are also described in this instantiation. The keep operator is used to express that the value of the listed attributes are kept unchanged. The value of an attribute will change non-deterministically in MAL, unless this is ruled out by the keep operator, or the value is explicitly set by the modal axiom.

The middle sentence of the manual extract is not directly included in the axiom in Figure 5. Because that sentence is about the red color, it is addressed in a different axiom, dealing with the red color.

2) Panels: Besides defining the values being displayed and the actions supported, each interactor representing a panel also models the color of the panel's access button and its blinking characteristic.

The TMT interactor is therefore described as follows (note that since its purpose is to provide the means to monitor variables there are no user actions relating to the variables in this case).

interactor TMT
aggregates
$tmtVariable$ **via** $BD1A$
$tmtVariable$ **via** $BD1B$
...
attributes
$characteristic : Characteristic$
$color : Color$
actions
$setValue2(Id, Int)$

The manual states that an access button is red and blinking when at least one critical variable of a given panel is in a non-acknowledged alarm state. These constraints are expressed as an invariant over the interactor as is illustrated by the following invariant that expresses the blinking red condition:

$$\begin{aligned}&BD1A.alarmState = NAckAlarm \rightarrow \\ &\qquad (color = red\ \&\ characteristic = blink) \\ &BD1B.alarmState = NAckAlarm \rightarrow \\ &\qquad (color = red\ \&\ characteristic = blink)\end{aligned}$$

The *setValue2* action is used to change the value of a variable on the panel. The first argument identifies the variable, the second its new value. It is coordinated with *BD1A.setValue* and *BD1B.setValue* (depending on the first argument) by adding constraints to ensure the requirement that when a variable sets its value the panel performs the same action (the effect operator asserts the occurrence of an action):

$$\begin{aligned}&\mathsf{effect}(BD1A.setValue(\_v)) \rightarrow \\ &\qquad \mathsf{effect}((setValue2(BD1A, \_v)) \\ &\mathsf{effect}(BD1B.setValue(\_v)) \rightarrow \\ &\qquad \mathsf{effect}(setValue2(BD1A, \_v)) \\ &\mathsf{per}\ (setValue2(BD1A, \_v)) \rightarrow \\ &\qquad \mathsf{effect}(BD1A.setValue(\_v)) \\ &\mathsf{per}\ (setValue2(BD1B, \_v)) \rightarrow \\ &\qquad \mathsf{effect}(BD1B.setValue(\_v))\end{aligned}$$

The first two implications specify that when the *BD1A* or *BD1B* variables set their values the setting of the variable is done in the telemetry panel too (i.e., *setValue2* happens when any of the variables are set). The third and fourth implications state that *setValue2* for *BD1A* and *BD1B* are only permitted when *BD1A* and *BD1B* respectively are set. In other words *setValue2* cannot happen unless a variable is set.

For the sake of simplicity the axioms above describe a panel with only two variables. Adding more variables amounts to adding more axioms, or adding conditions to the existing ones using the same principle. It should be noted that in the final

$$
\begin{aligned}
[setValue(\_v)] \quad & (\quad (\ (critical \ \& \ (\ (\_v >= infAlarmLim \ \& \ \_v < infAlertLim) \\
& \qquad\qquad\qquad\qquad \mid (\_v <= supAlarmLim \ \& \ \_v > supAlertLim))) \\
& \qquad\qquad \mid (!critical \ \& \ ((\_v < infAlarmLim) \mid (\_v > supAlarmLim)))) \\
& \qquad \& \ (alarmState \ != AckAlarm \ \& \ alarmState \ != NAckAlarm)) \\
\rightarrow \quad & value' = \_v \ \& \ color' = yellow \ \& \ error' = Lim \\
& \& \ alertState' = NAckAlert \ \& \ characteristic' = Blink \\
& \& \ \mathsf{keep}(supAlertLim, infAlertLim, supAlarmLim, infAlarmLim, unit, critical, alarmState)
\end{aligned}
$$

Fig. 5. Axiom for the blinking yellow case ($tmtVariable$ interactor)

version of the model simplification was achieved by combining some of these axioms.

3) Adding more panels: The other panels defined for the system have similar characteristics. For example, the SIN panel features safety relays in addition to variables. Each relay indicates whether it can be released or not. Relays are defined as boolean attributes of the *SIN* Interactor. The SIN panel also has the additional feature that it can be set to automatic mode.

As long as no dependencies between the panels exist, adding more panels to the model is only a matter of aggregating their corresponding interactors in $main$. When interdependencies exist, the states and/or actions of the relevant interactors must be coordinated as will be discussed in the next section.

D. Coordination between the panels

The same variable can be represented in more than one panel thereby creating interdependencies between the panels. If the value changes in one panel, it must also change in the other. Examples are the alarms and alerts panels, which feature lists of alarms and alerts fed with information from the other panels. The approach used in the *TMT* panel is used here. The alerts and alarms variables are defined and aggregated to create the panels' models, as captured in the *ALER* and *ALAR* interactors. A distinctive feature of these panels is that their content is solely defined by the alarms and alerts occurring in the other panels in the system, and that the acknowledgment of alarms/alerts must also be reflected in the original variable. To support this an acknowledge action was added to the other interactors (e.g., *TMT* and *SIN*).

Coordination of variables' values between panels is expressed in the *main* interactor. This coordination can either be expressed in terms of invariants or in terms of modal axioms. The use of invariants has already been illustrated when specifying the coordination between the *TMT* panel and its internal variables. Invariants can also be expressed over two or more variables in different panels. To do this the axioms must be placed in the $main$ interactor. For example, specifying that the variable $BD1A$ must have the same value in the $monitTMT$ and $monitSIN$ interactors, is as follows:

$$monitTMT.BD1A.value = monitSIN.BD1A.value$$

Each interactor must then have (at least) one action that enables setting the value so that when it changes in one interactor it also changes in the other. The action being used, however, is not defined explicitly. While this mechanism for coordination scales well, it can generate unwanted side effects. Two different axioms might, for example, express requirements over attributes that, although not contradictory, mean that no actions exist that achieve both effects. This will create a deadlock in the model (i.e., a situation where no action is possible) whose cause can be hard to diagnose. Auxiliary actions must be added to solve the problem which can increase complexity.

An alternative approach expresses coordination directly using modal axioms in the definition of actions. This can be done at the level of actions or at the level of attributes. At the level of actions, the coordinating main $setValue$ action invokes the actions in the coordinated panels:

$$
\begin{aligned}
[setValue(id, \_v)] \ id = BD1Aid \ \rightarrow \\
\mathsf{effect}(monitTMT.setValue(BD1Aid, \_v)) \\
\& \ \mathsf{effect}(monitSIN.setValue(BD1Aid, \_v))
\end{aligned}
$$

At the level of attributes, the coordinating axiom, explicitly sets the attributes in the coordinated panels:

$$
\begin{aligned}
[setValue(id, \_v)] \ id = BD1Aid \ \rightarrow \\
monitTMT.BD1A.value' = \_v \\
\& \ monitSIN.BD1A.value' = \_v
\end{aligned}
$$

In either case, setting the values in the panels must be constrained to happen only when $setValue$ happens in the $main$ interactor, otherwise coordination would not be guaranteed:

$$
\begin{aligned}
&\mathsf{per} \ (monitTMT.setValue(\_id, \_v)) \ \rightarrow \\
&\qquad\qquad \mathsf{effect}(setValue(\_id, \_v)) \\
&\mathsf{per} \ (monitSIN.setValue(\_id, \_v)) \ \rightarrow \\
&\qquad\qquad \mathsf{effect}(setValue(\_id, \_v))
\end{aligned}
$$

The example illustrates that while this formulation makes the coordination rationale somewhat more explicit, it is more complex to formulate. In addition it does not completely eliminate the implicit execution of actions. As a result of this exercise an approach based on invariants was favored.

E. Confidence in the model

Before verification of the model can proceed, confidence that the model serves as the required reference model must be established. Confidence can be established by appealing to the process that was used to construct the model. The User Operation Manual contains a detailed description of the interface and its operation. This information was systematically translated into the model as has already been illustrated. The modelers' understanding of the manual was checked through discussion with a subject matter specialist from the IAE. The model was based on our understanding of what the user interface should

be. At the end of the process, the model was checked against the manual by a member of the modeling team.

Confidence in the model was further achieved by investigating its behavior. This involved stating that some goal of the system cannot be achieved and checking template properties. This process aims to get counter-examples that can then be analyzed with subject matter specialists to check whether the illustrated behavior is consistent with understanding of the system. Examples of how this was done are presented in the next section.

When two axioms define conflicting behaviors for an action, the model becomes empty and anything can be proved true of its behavior. Although the model checker (NuSMV) can be set to check for transition relation totality and the absence of deadlocks, the negation of some of the initially proved properties was also checked as a means to prevent any problems at this level.

VI. Analysis

The first step in the analysis was to formulate properties expressed in CTL. Once a property has been formulated it is checked using the model checker, and the result interpreted. Checking the property is an automated step. However, formulating properties and interpreting results, although tool supported, requires human judgment.

At this point it must be stressed that the goal of the analysis if not to prove the system *correct*. What the analysis provides is an exhaustive investigation of the model against a collection of properties found relevant of the system under consideration. Two sources of properties can be identified. The first is knowledge of the domain. Clearly domain knowledge plays a relevant role in identifying relevant properties that a particular user interface should or should not exhibit. The other regards intrinsic properties of the user interface that are commonly considered as good practice (for example, Nielsen's usability heuristics [38]). For both cases, IVY provides a collection of property specification patterns to help identify and express relevant properties (see Section IV-C above).

A. Formulating properties

The patterns provided by the IVY tool were used to generate properties for verification. An often-cited user-related requirement of such a safety critical system (**feedback**) will be used to illustrate the process. Providing appropriate feedback is important in maintaining situation awareness and avoiding mode confusion problems. Feedback is best considered in the context of a system's goals. An example is whether feedback is always provided to the operator when a variable generates an alarm or alert. For example, in the case of the telemetry panel and the displayed attribute *BD1A* (*monitTMT.BD1A*), the instantiation of the feedback pattern is presented in Figure 6. The instantiation specifies that the alarm/alert state is represented by the *error* attribute. Feedback is provided by changing the color of the telemetry panel's access button (modeled by attribute *monitTMT.colour*).

A family of *IVAL1*-indexed properties is generated from the template:

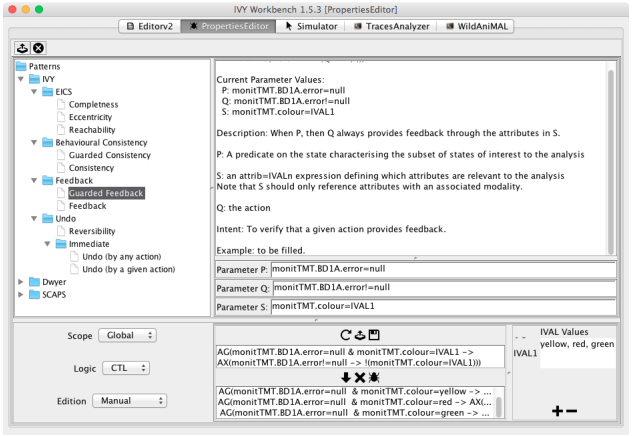

Fig. 6. The feedback pattern's instantiation

$$AG(monitTMT.BD1A.error = null \;\&\; monitTMT.colour = IVAL1 \rightarrow AX(monitTMT.BD1A.error\ != null \rightarrow\ !(monitTMT.colour = IVAL1)))$$

where *IVAL1* ranges over the possible colors for the button (yellow, red and green). This will generate three properties, one for each value of *IVAL1*. Note that these properties do not have to be written, they are automatically generated by successive instantiation of values of *IVAL1*.

B. Verification

One interesting example of a property used in establishing confidence in the model is presented in Property 1.

$$AG(monitTMT.BD1A.colour = green \rightarrow\ !EX(monitTMT.BD1A.colour = red)) \quad (1)$$

To check that variables can be in an alert state, the property negates that possibility. For variable *BD1A*, the property states that for all states (*AG*) where the variable is in normal state (its color is green: *monitTMT.BD1A.colour* = *green*), there is no next state (*!EX*) where the variable is in an alarm state (color red: *monitTMT.BD1A.colour* = *red*):

The property proved to be false, and a trace was generated (see Figure 3). This trace highlighted a situation where the *BD1A* variable is red under an *acknowledged* alert condition. This situation conflicted with an understanding of how the system worked based on the manual. Analysis of the model indicated a lack of specification of what happens to a non-critical alert. The manual on which the model was based did not treat these conditions accurately. In practice this only represents a problem if it is possible for a non-critical alert to occur, and as a result the operator could be surprised by this unexpected and undocumented behavior. This analysis enabled us to uncover the potential for an automation surprise. This finding led to discussion with the operator of the system and the model was updated to consider non-critical alerts.

The *feedback* template, using the instantiation described in the previous section, generates three properties:

$$AG(monitTMT.BD1A.error = null \quad (2)$$

$$\begin{aligned} & \& \ monitTMT.colour = yellow \\ & \rightarrow \ AX(monitTMT.BD1A.error \ != null \\ & \qquad \rightarrow \ !(monitTMT.colour = yellow))) \end{aligned}$$

$$\begin{aligned} & AG(monitTMT.BD1A.error = null \qquad (3) \\ & \quad \& \ monitTMT.colour = red \\ & \quad \rightarrow \ AX(monitTMT.BD1A.error \ != null \\ & \qquad \rightarrow \ !(monitTMT.colour = red))) \end{aligned}$$

$$\begin{aligned} & AG(monitTMT.BD1A.error = null \qquad (4) \\ & \quad \& \ monitTMT.colour = green \\ & \quad \rightarrow \ AX(monitTMT.BD1A.error \ != null \\ & \qquad \rightarrow \ !(monitTMT.colour = green))) \end{aligned}$$

The properties hold when the color is green or red (Properties 3 and 4), but not when it is yellow (Property 2). A counter-example indicates that the button may be yellow already because another variable is being alerted. The button's color, on its own, is not enough to guarantee that feedback is provided when new alerts are raised. In fact even the beep feature does not solve the problem because that feature can be turned off. This is not necessarily a problem, but raises discussion about the level of feedback provided by the system. Is immediate feedback about individual variables needed, or is the goal to call attention to alarms and alerts in general? Discussion with operators clarified that the goal is to generate a general call to attention.

It is clear that properties being proved and results of proofs must always be interpreted in the context of the broader system and its requirements and with a sense of the future versions of the system. For example (also in the light of the results for property 2) consider whether property 3 being true is a positive result. If an alarm is issued, surely the button should remain red. Looking more closely, this two-variable model shows that BD1A is the only critical variable in the TMT interactor. This means that the color can only be red when BD1A is in an error state, as demonstrated by the fact that Property 5 below fails. Checking for models with a larger number of variables indicates the same situation relating to yellow.

$$\begin{aligned} & EF(monitTMT.BD1A.error = none \qquad (5) \\ & \qquad \& \ monitTMT.colour = Red) \end{aligned}$$

A second requirement that was analyzed was whether the help panel is always available to the operator. Which panel is presented to the user at any given moment is modeled by the *display* attribute in the main interactor. A total of 23 panels were considered, although only the content of the Synoptics, Telemetry, Alarms and Alert panels were modeled in detail. For the others, the *display* attribute models the fact they are being displayed and the behavior of the system is constrained accordingly. The help panel is modeled by the value *Help*. The action used to access the panel is action *helpBtn*. This requirement was checked through successive instantiations of the **universality** pattern. This pattern captures the requirement that some condition on the state of the system always holds. Instantiating the pattern explored whether the *helpBtn* action always implies that the *Help* panel is displayed (*display=Help*). This was expressed in Property 6, which verified true.

$$AG(effected(helpBtn) \ \rightarrow \ display = Help) \qquad (6)$$

Care must be taken in interpreting what has been proved. Property 6 expresses that in all states, if the *helpBtn* action is executed, then the help panel will be displayed. This, however, is not the same as saying that the help panel is always reachable using the *helpBtn* action. In fact the *helpBtn* action is not always available.

Two possibilities avoid this. The first is to check simply whether the help panel is always reachable (Property 7) but this says nothing about how it is reached. The other is to check explicitly that the action is available in all states. With this approach, Property 6 becomes Property 8.

$$AG(EX(display = Help)) \qquad (7)$$

$$\begin{aligned} & AG(EX(\quad effected(helpBtn) \qquad (8) \\ & \qquad \& \ (effected(helpBtn) \ \rightarrow \ display = Help))) \end{aligned}$$

Both formulations of the property fail indicating that help is not possible from the access panel (the initial state). Further possibilities can be explored by removing the access panel from consideration:

$$\begin{aligned} & AG(display \ != SysAccess \qquad (9) \\ & \quad \rightarrow \ EX(effected(helpBtn) \ \& \ display = Help)) \end{aligned}$$

This property also fails. The property can be generalized to all panels by introducing a variable ranging over the set of all panels, generating 23 properties:

$$\begin{aligned} & AG(display = IVAL1 \qquad (10) \\ & \quad \rightarrow \ EX(effected(helpBtn) \ \& \ display = Help)) \end{aligned}$$

A total of five panels that had no access to a help system were found as a result: the lock panel, the sequence of panels that the operator must go through during login, and the logout panel.

Figure 7 presents the relevant excerpt (the attributes and action for the *main* interactor) of the trace for the logout panel (*EndScr*) case. The trace ends at state 5 with *display=Final*. The fact that there is no next state points out the fact that the *helpBtn* action is not possible in that panel. This issue was discussed with domain (and, possibly, usability) experts who came to the conclusion that in this particular case the situation was sensible. This analysis demonstrates that it is possible to explore all states of the system's user interface systematically, and identify deviations from expected (or expressed) behavior.

Further analysis considered the consistency of the user interface, for example in the acknowledgment of panels, and the reversibility of user actions. Patterns were instantiated as described above.

VII. Discussion

This analysis has shown how more complex display structures can be modeled and analyzed using the IVY tool. In this particular case the model was developed from the User Operation Manual, while interaction with the IAE team was kept to a minimum. The model therefore represents the information that is provided to operators as a guide rather than necessarily reflecting the actual implementation. Proving properties can therefore be understood as investigating the quality of the provided information. The analysis made it

	1	2	3	4	5
main.action		enterBtn	executeBtn	executeBtn	exitBtn
beepEnbld	FALSE	FALSE	FALSE	FALSE	FALSE
beeping	FALSE	FALSE	FALSE	FALSE	FALSE
display	SysAccess	SelScr	SEVaccess	MainScr	EndScr
monitALAR.BD1A.action		setState(NAckAlarm)			
critical	TRUE	TRUE	TRUE	TRUE	TRUE
error	none	Lim	Lim	Lim	Lim
state	Good	NAckAlarm	NAckAlarm	NAckAlarm	NAckAlarm

Fig. 7. Trace illustrating unavailability of the *help* action in the *Final* panel (excerpt)

possible to identify instances where not enough information is provided by the manual. We argue that this lack of appropriate information is relevant on two accounts. On the one hand, it might lead to automation surprises during the operation of the system. The model that the operator is able to build from the manual is incomplete. This is particularly relevant in systems such as the one being analyzed where operating procedures are followed, and little or no exploration of the systems outside these procedures is carried out. On the other hand, if we consider the operator manual as the *de facto* specification of the user interface, then the fact that it is incomplete will have a negative impact on the design and testing of updates or future versions of the system.

The considered requirements were properties derived from the IVY templates. The failure of properties, and also their success, prompted discussion with operators about, for example, the adequacy of alert and alarm conditions as well as the availability of on-line help. While we make no claim that the set of proved properties is exhaustive, the paper has demonstrated the feasibility of proving sets of externally defined requirements as discussed elsewhere in [23]. It also demonstrates that a reference model can be constructed that can be used to analyze future versions of the design.

An additional issue relevant to this exercise was the scaleability of the approach. While the paper describes the simpler versions of the models (considering very few variables), the complete study involved adding more detail to the model. Building the larger model involved just adding more variables and corresponding axioms and involved no particular complexity. The model with four panels (Telemetry, Synoptic, Alarms and Alerts), with 4 system variables (two represented through aggregation and two through attributes), plus all the navigation logic, amounts to 490 lines of MAL (including comments). More relevantly, the model consists of 9 interactors, totaling 168 axioms (of which 4 are initialization axioms, 52 are permissions, 64 are modal, and the remaining 48 are invariants) producing a state space of approximately $8e^6$ reachable states. Although exact numbers will depend on the concrete type of variable being added, the size of the model grows roughly linearly with every new variable added (around an extra 50 lines of MAL code, with 7 to 8 new permission and modal axioms, and the number of interactors remaining constant).

Verification times are dependent on a number of factors, including the concrete property being checked and the ordering of variables used. While experimenting with the two approaches to expressing coordination, we found that invariants make the model checker consume more memory during the verification, as well as taking more time to complete the analysis. Nevertheless, without any particular attempt to set up the verification process for speed, the properties in the previous section were all verifiable in under five minutes. The addition of new variables substantially increases verification time, but by adjusting the model checker's configuration it was still possible to check larger models in reasonable time. Even though adding more variables increases verification time quickly, adding more variables does not necessarily add more relevant information to the model. In a parallel piece of work we are currently comparing model checking as a technology with theorem proving in terms of time taken for the automated analysis vs. time and difficulty for the human analyst to perform the proof [23].

The model described in the paper together with a model of another subsystem and a number of alternative and complementary variants were developed in the context of a masters dissertation in software engineering over a period of 6 months. A considerable part of this time involved study of the user manual and of the alternative modeling approaches. A final validation of the model by the first author, was done in the space of a week. The student (Manuel Sousa) had a background in formal methods, part of the MSc curriculum, but no extensive experience of formal methods usage in a system of the scale of this one. He had no background in Human-Computer interaction. While formal methods use is still not widespread and requires appropriate expertise, tools such as the IVY workbench aim to ease the learning curve for specific application domains. Results show that, with reasonable knowledge of formal methods (such as obtained at MSc level) and adequate guidance, an approach such as the one described can be used to identify potential problems in the user interface of a complex safety critical system. Additionally, they show that the analysis can be carried out independently, based on a description of the system under analysis and using interaction with domain experts to validate their understanding of the provided documentation (effectively hiding the technical details of the approach), and still provide relevant results.

If a model is to be used as a basis for demonstrating that requirements are true of a device, it is necessary to demonstrate that the model is a faithful description of the device. One way of doing this is to develop the device formally by refining the model as supported by tools such as Event B [43]. Alternatively a model could be generated from the code of an existing device. This can be achieved by using a set of rules to generate a model from code as discussed

in [44], and is an approach we are exploring [45] (see also [46]). In the approach taken here, confidence is grounded on on a two fold argument. First, the fact that the user operation manual can be considered as the *de facto* specification of the system's user interface; second, the systematic translation of such documentation into the modeling notation, and the comparison of the behavior exhibited by the model against the expected behavior of the system. The possibility of validating the model through simulation is currently under development.

VIII. Conclusions

Formal verification techniques have an important role in reducing risk in safety-critical interactive systems. The model checking based approaches used in this paper have the rigor, and the tool support needed to perform exhaustive and automatic verification of key properties of system designs. The approach also has potential to prove systematically that capabilities are supported.

The paper has described how the IVY workbench was applied to an existing aerospace system. Unlike previous work, the model was based solely on a description of the system, making the exercise closer to that of applying the approach during development. The size and complexity of the system also meant that a more structured approach to modeling the different panels in the user interface, and (especially) their coordination had to be developed.

Our experience has shown that the proposed approach provides a practical and feasible way to systematically specify, and automatically verify, the user interfaces of complex systems. In most of the cases, such verification activities are currently restricted to inspections of documents and test results analysis, where the interpretation of the results can still be quite subjective and the test scenarios may not cover all the possible combinations of actions that can take place. The support of a computer-aided tool would greatly expedite the accomplishment of this activity. The overall approach described in this paper represents a significant step forward.

Acknowledgment

The authors would like to thank the anonymous reviewers for helpful feedback on the paper.

References

[1] D. MacKenzie, "Computer-related accidental death: an empirical exploration," *Science and Public Policy*, vol. 21, no. 4, pp. 233–248, 1994.

[2] J. Rushby, "Using model checking to help discover mode confusions and other automation surprises," *Reliability Engineering and System Safety*, vol. 75, no. 2, pp. 167–177, February 2002.

[3] J. C. Campos and M. D. Harrison, "Systematic analysis of control panel interfaces using formal tools," in *Interactive systems: Design, Specification and Verification, DSVIS '08*, ser. Springer Lecture Notes in Computer Science, N. Graham and P. Palanque, Eds., no. 5136. Springer-Verlag, 2008, pp. 72–85.

[4] ——, "Interaction engineering using the IVY tool," in *ACM Symposium on Engineering Interactive Computing Systems (EICS 2009)*. New York, NY, USA: ACM, 2009, pp. 35–44.

[5] M. Harrison, J. Campos, and P. Masci, "Reusing models and properties in the analysis of similar interactive devices," *Innovations in Systems and Software Engineering*, 2013.

[6] J. C. Campos, G. Doherty, and M. D. Harrison, "Analysing interactive devices based on information resource constraints," *International Journal of Human-Computer Studies*, vol. 72, pp. 284–297, 2014.

[7] M. Sousa, J. Campos, M. Alves, and M. Harrison, "Formal verification of safety-critical user interfaces: a space system case study," in *Formal Verification and Modeling in Human Machine Systems: Papers from the AAAI Spring Symposium*. AAAI Press, 2014, pp. 62–67.

[8] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, April 1986.

[9] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements specification for process-control systems," *IEEE Trans. Soft. Eng.*, vol. 20, no. 9, pp. 684–706, September 1994.

[10] F. Schneider, S. M. Easterbrook, J. R. Callahan, and G. J. Holzmann, "Validating requirements for fault tolerant systems using model checking," in *Proc. 3rd Interntional Conference on Requirements Engineering*, Colorado Springs, Colorado, April 1998, pp. 4–13.

[11] K. Havelund, M. Lowry, and J. Penix, "Formal analysis of a space craft controller using SPIN," *IEEE Transactions on Software Engineering*, vol. 27, no. 8, pp. 749–765, 2001.

[12] P. R. Glück and G. J. Holzmann, "Using SPIN model checking for flight software verification," in *Proc. IEEE Aerospace Conference*, Big Sky, Montana, March 2002, pp. 105–113.

[13] M. C. B. Alves, D. Drusinsky, J. B. Michael, and M. Shing, "Formal validation and verification of space flight software using statechart-assertions and runtime execution monitoring," in *Proc. 6th IEEE International Systems of Systems Conference*, Albuquerque, N.M., June 2011, pp. 155–160.

[14] J. C. Campos and M. D. Harrison, "Model checking interactor specifications," *Automated Software Engineering*, vol. 8, no. 3-4, pp. 275–310, August 2001.

[15] M. C. Elder and J. C. Knight, "Specifying user interfaces for safety-critical medical systems," in *2nd Annual Intl. Symp. Medical Robotics and Computer Assisted Surgery*. Wiley-Liss, 1995, pp. 148–155.

[16] P. Masci, R. Ruksenas, P. Oladimeji, A. Cauchi, A. Gimblett, Y. Li, P. Curzon, and H. Thimbleby, "On formalising interactive number entry on infusion pumps," *Electronic Communications of the EASST*, vol. 45: Formal Methods for Interactive Systems 2011, 2011.

[17] M. L. Bolton, E. Bass, and R. Siminiceanu, "Using formal verification to evaluate human-automation interaction, a review," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, no. 99, pp. 1–16, 2013.

[18] M. Feary, "Automatic detection of interaction vulnerabilities in an executable specification," in *Proceedings of the 7th International Conference on Engineering Psychology and Cognitive Ergonomics*, ser. EPCE'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 487–496.

[19] M. L. Bolton, "Validating human-device interfaces with model checking and temporal logic properties automatically generated from task analytic models," in *Proc. 20th Behav. Represent. Model. Simul. Conf.*, 2011, pp. 130–137.

[20] G. Lüttgen and V. Carreño, "Analyzing mode confusion via model checking," in *Proc. Theor. Pract. Aspects SPIN Model Check*, 1999, pp. 120–135.

[21] A. Joshi, S. P. Miller, and M. P. E. Heimdahl, "Mode confusion analysis of a flight guidance system using formal methods," in *Proc. 22nd Digit. Avionics Syst. Conf.*, Oct 2003, pp. 2.D.1–1–2.D.1–12.

[22] P. Masci, A. Ayoub, P. Curzon, M. Harrison, I. Lee, O. Sokolsky, and H. Thimbleby, "Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example," in *Proceedings ACM Symposium Engineering Interactive Systems (EICS 2013)*. ACM Press, 2013, pp. 81–90.

[23] M. D. Harrison, P. Masci, J. Campos, and P. Curzon, "Demonstrating that medical devices satisfy user related safety requirements," in *Proceedings of Fourth Symposium on Foundations of Health Information Engineering and Systems (FHIES) & Sixth Software Engineering in Healthcare (SEHC) Workshop*. Springer-Verlag, 2014, accepted.

[24] B. Buth, "Analyzing mode confusion: An approach using FDR2," in *Proc. 23rd Int. Conf. Comput. Safety, Rel., Security*, 2004, pp. 101–114.

[25] J. Bredereke and A. Lankenau, "Safety-relevant mode confusions – modelling and reducing them," *Reliability Engineering & System Safety*, vol. 88, no. 3, pp. 229 – 245, 2005.

[26] P. Palanque, R.Bastide, and V. Senges, "Validating interactive system design through the verification of formal task and system models," in *6th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'95)*, Wyoming, U.S.A., August 1995.

[27] F. Paternò and C. Santoro, "Preventing user errors by systematic analysis of deviations from the system task model," *International Journal of Human-Computer Studies*, vol. 56, no. 2, pp. 225–245, February 2002.

[28] M. Bolton, R. Siminiceanu, and E. Bass, "A systematic approach to model checking human–automation interaction using task analytic models," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 41, no. 5, 2011.
[29] A. Blandford, R. Butterworth, and P. Curzon, "Models of interactive systems: a case study on programmable user modelling," *International Journal of Human-Computer Studies International Journal of Human-Computer Studies*, vol. 60, pp. 149–200, 2004.
[30] B. E. John, K. Prevas, D. D. Salvucci, and K. Koedinger, "Predictive human performance modeling made easy," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '04*. ACM, 2004, pp. 455–462.
[31] R. Rukšėnas, J. Back, P. Curzon, and A. Blandford, "Verification-guided modelling of salience and cognitive load," *Formal Aspects of Computing*, vol. 21, pp. 541–569, 2009.
[32] J. Berstel, S. Reghizzi, G. Rouseel, and P. Pietro, "A scalable formal method for the design and automatic checking of user interfaces," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 2, pp. 124–167, 2005.
[33] M. Dwyer, G. Avrunin, and J. Corbett, "Property Specification Patterns for Finite-State Verification," in *2nd Workshop on Formal Methods in Software Practice*, M. Ardis, Ed., March 1998, pp. 7–15.
[34] M. L. Bolton, N. Jiménez, M. M. van Paassen, and M. Trujillo, "Automatically generating specification properties from task models for the formal verification of human-automation interaction," *IEEE Transactions on Human-Machine Systems*, vol. 44, no. 5, pp. 561–575, October 2014.
[35] D. J. Duke and M. D. Harrison, "Abstract interaction objects," *Computer Graphics Forum*, vol. 12, no. 3, pp. 25–36, 1993.
[36] D. Ramos-Hernandez, P. Fleming, and J. Bass, "A novel object-oriented environment for distributed process control systems," *Control Engineering Practice*, vol. 13, no. 2, pp. 213 – 230, 2005.
[37] M. Dwyer, G. Avrunin, and J. Corbett, "Patterns in property specifications for finite-state verification," in *21st International Conference on Software Engineering, Los Angeles, California*, May 1999, pp. 411–420.
[38] J. Nielsen, *Usability Engineering*. Academic Press, Inc., 1993.
[39] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An Open Source Tool for Symbolic Model Checking," in *Computer-Aided Verification (CAV '02)*, ser. Lecture Notes in Computer Science, K. G. Larsen and E. Brinksma, Eds. Springer-Verlag, 2002, vol. 2404.
[40] J. Campos and M. Harrison, "Modelling and analysing the interactive behaviour of an infusion pump," *Electronic Communications of the EASST*, vol. 45: Formal Methods for Interactive Systems 2011, 2011.
[41] M. Fowler, *UML Distilled: a brief guide to the standard object modelling language*, 3rd ed., ser. The Addison-Wesley Object Technology Series. Addison-Wesley, 2004.
[42] M. A. Barbosa, L. S. Barbosa, and J. C. Campos, "A coordination model for interactive components," in *Fundamentals of Software Engineering*, ser. Lecture Notes in Computer Science, vol. 5961. Springer-Verlag, 2010, pp. 416–430.
[43] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
[44] G. J. Holzmann, "Trends in software verification," in *FME 2003: Formal Methods*, ser. Lecture Notes in Computer Science, K. Araki, S. Gnesi, and D. Mandrioli, Eds. Springer Berlin Heidelberg, 2003, vol. 2805, pp. 40–50.
[45] J. C. Silva, J. C. Campos, and J. A. Saraiva, "GUI inspection from source code analysis," *Electronic Communications of the EASST*, vol. 33: Foundations and Techniques for Open Source Software Certification 2010), 2010.
[46] A. Gimblett and H. Thimbleby, "User interface model discovery: Towards a generic approach," in *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '10)*. New York, NY, USA: ACM, 2010, pp. 145–154.

José Creissac Campos is an Assistant Professor at the Department of Informatics of the University of Minho, and a senior researcher at HASLab/INESC TEC, in Braga, Portugal. His research focuses on the application of software engineering techniques and tools to the modeling and analysis of interactive systems, aiming at bringing closer the software engineering and human-computer interaction (HCI) fields. Besides leading the development of the IVY workbench, current and recent funded research includes applying model-based testing to user interfaces, prototyping ambience intelligence systems using virtual reality simulations, and reverse engineering user interfaces.

Manuel Sousa holds an MSc. in Software Engineering from the University of Minho. He is currently a User Experience consultant at WeDo Technologies.

Miriam C. Bergue Alves received the Doctoral degree in applied computer science from the National Institute for Space Research, São José dos Campos, Brazil, in 1999. She is a Government Researcher with the Institute of Aeronautics and Space at the Department of Aerospace Science and Technology, São José dos Campos, Brazil, and during 2010 and 2011 she was a Post-Doctoral Researcher with the Department of Computer Science, Naval Postgraduate School, Monterey, California. Her research interests include modeling, design, reliability and formal V&V of mission-critical systems. She has been developing space and aeronautical software systems since 1995 and has leaded the team responsible for flight software development of the Brazilian Satellite Launcher Program.

Michael D. Harrison is Emeritus Professor of Informatics and senior research investigator at Newcastle University and research fellow at QMUL (funded to work on the analysis of medical devices). His research focuses on the systematic analysis of the functional behavior of interactive systems using a combination of model checking and automated theorem proving techniques. These techniques have also been used to model human aspects of ubiquitous systems with the aim of enabling a larger scale analysis of the properties of implicit actions that can take place within a physical environment augmented by technology.