Interaction Engineering Using the IVY Tool

José C. Campos
Department of Informatics/CCTC
Universidade do Minho
4710-057 Braga, Portugal
+351 253 60 4447
jose.campos@di.uminho.pt

Michael D. Harrison
School of Computing Science, Newcastle University
Claremont Tower
Newcastle upon Tyne, NE1 7RU, UK
+44 191 222 8218
michael.harrison@ncl.ac.uk

ABSTRACT
This paper is concerned with support for the process of usability engineering. The aim is to use formal techniques to provide a systematic approach that is more traceable, and because it is systematic, repeatable. As a result of this systematic process some of the more subjective aspects of the analysis can be removed. The technique explores exhaustively those features of a specific design that fail to satisfy a set of properties. It also analyzes those aspects of the design where it is possible to quantify the cost of use. The method is illustrated using the example of a medical device. While many aspects of the approach and its tool support have already been discussed elsewhere, this paper builds on and contrasts an analysis of the same device provided by a third party and in so doing enhances the IVY tool.

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification – *model checking.* D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *theory and methods.*

General Terms
Design, Reliability, Human Factors, Verification.

Keywords
Formal Methods, model-based usability analysis.

1. INTRODUCTION
The aim of usability analysis is to predict problems that might be barriers to the use or effectiveness of an interactive device early in the design process. By being effective in the early stages of design it can forestall the costs of redesign. This paper is concerned with formal techniques that provide early feedback about the design of an interactive system before expensive decisions have been made. Usability analysis is usually intended for use by human computer interaction experts. They operate on an informal representation of the design, which can be more readily changed, for example a storyboard or a draft of the user manual. A primary aim of this analysis is to achieve usability improvement while at the same time minimizing the cost of application.

Because typical usability evaluation methods are informal it is sometimes difficult and tedious to perform them systematically and accurately. They rely on application by those who have sufficient expertise to apply them correctly, for example interpreting any questions that are asked of the design appropriately. Although the aim of these methods is that they should be cheap to apply, the cost and the benefit of application depends on the expertise of the analyst. A common concern is that the influence of expert judgment is such that the evaluation of a specific device is not repeatable [2,11].

This paper explores the process and tool support for a particular formal approach to systematic analysis and extends work reported in [5]. While the approach has already been discussed in the context of an automobile air conditioning system, the paper explores the whole process of usability evaluation in more detail using the IVY tool. It also recognizes and accommodates properties of a device that relate to user effort. These properties were used to analyze the same device in [21]. To repeat the analysis performed in [21], new property patterns were incorporated into the IVY tool relevant to the additional properties checked. This comparative analysis justifies the use of a relatively small example. However it should be said that there is a large class of designs of interactive devices similar in complexity to this example, and evidence shows they are not free from usability problems.

While the focus of the analysis performed using the IVY tool in [5] was to find usability problems that arise because a property akin to a usability heuristic is broken, the comparable analysis of [21] on the same device focuses on a different kind of issue. That analysis is concerned with the "cost" of using an interface. It is argued in [21] that cost can be addressed by looking at the model as a graph and considering user effort as defined in terms of the number of steps that have to be taken to follow different kinds of path within the graph. This paper applies both styles of analysis and, as a result, enhances the process originally discussed.

The paper makes a number of contributions:

- It revisits the analysis process proposed in [5] and describes the complete architecture of the IVY tool designed to support it. The IVY tool makes the process of usability evaluation more repeatable and therefore brings more objectivity to the evaluation process.
- It expands on the type of reasoning that can be supported by IVY. To this end, a number of new property patterns are introduced.

EICS'09, July15–17, 2009, Pittsburg, Pennsylvania, USA.

- It applies the tool to a model and verification requirements defined by a third party. This illustrates the flexibility and power of the tool, and represents an effort to repeat and thereby validate an analysis done elsewhere.

2. RELATED WORK

Most of the techniques aimed at identifying usability requirements or applying usability analysis have relatively well elaborated processes. For example the stages of cognitive walkthrough [19] and heuristic evaluation [17] are clear described and adhered to by analysts using the techniques. Relatively few of these techniques however benefit from tool support. Even in the context of techniques of human reliability assessment (which have strong affinities with the techniques described here) where evidence of the dependability of the system is a primary focus, few tools have been developed that support the whole process. Consider the examples of ATHEANA [8] and CREAM [13], the ability to trace the application of the technique and to provide evidence of how the method is applied is an important requirement of regulators.

The focus of the IVY tool is analysis and central to this analysis is a model checking approach [7]. Hence key features of the design are concerned with aiding the construction of properties based on usability patterns and the analysis of the traces that are generated as counter examples when the properties break. For this reason a notation was used with a precise semantics to make it relatively straightforward to map to a standard model checking tool. Previous work [14] has used statecharts with much more complex semantics. Model checking has been used elsewhere in the analysis of user interfaces, for example [1]. The difference here is to use model checking as the core of an approach to the systematic analysis of interactive systems.

Those tools that do exist in human computer interaction are more concerned with representation than with analysis. Hence Petshop [16] is concerned to aid the development and animation or prototyping of ICO based models. There are also a number of tools designed to support the construction of tasks (see, for example [18]). In general these tools are not designed to support usability analysis as such, rather to aid the organization and structuring of a model of the design of the interactive system. These approaches use formal notations to specify interaction or task to map to implementation or to provide animations of the design. The animations in [16,18] are based on Petri nets or CTT models, the animation in this work is based on the traces that are generated as counterexamples when properties break.

3. THE PROCESS

Analysis can be performed at different points in the development process, and with different purposes. The kind of model used varies with the type and purpose of the analysis. The concern in the IVY process is formative, to produce design feedback at the early stages of development. The goal is that models that capture specific aspects of the system might be quickly developed and analyzed using tool support (see Figure 1).

Four important representations are described in this process. The *model* captures key features of the device. It is developed at a level of abstraction that emphasizes these features while at the same time making it possible to reason about the system, providing an analysis that will form the basis for the design. A *trace* is a sequence of states that (for these purposes) breaks a property and forms the basis for the analysis. The trace is generated by the model checker that underlies the IVY tool. Models and traces are directly related to two other representations. The *prototype* reflects the characteristics of the model and can be used to explore what the system is like. This can be achieved producing a more or less functional version of the device. The *scenario* is a narrative describing a context in which the device is used based on the trace. Hence the prototype and the scenario can be seen as bringing context and texture to the more abstract models and traces. Tool support can be used to extract a model from a prototype or to develop a prototype from the model. This makes it easier to move between levels of abstraction.

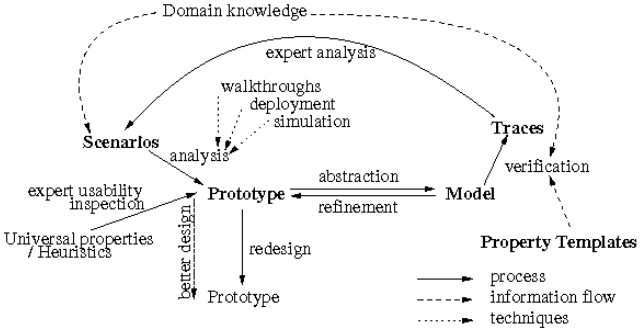

Figure 1. Analysis in the development process (adapted from [12])

A collection of properties are used to analyze the device. These properties are based on standard patterns and are applied systematically to the design. When a property cannot be verified, a device behavior is returned illustrating a behavior that made the property false. This behavior is described using a trace. It identifies a potential problem and must then be examined for plausibility. In this case cognitive plausibility plays a major role. The results of this examination can lead to the design of the model being refined if a genuine problem is identified. Alternatively it could highlight a flaw in the model (for example, inadequacy of abstractions used), or a possible but not relevant behavior (maybe because it is not cognitively plausible).

These representations and properties are combined using the IVY process. The designer produces a model based on a design. In the early stages of design this model could be based on a storyboard reflecting anticipated use. The model might be animated by the developer to obtain a look and feel of the envisaged design. This animation could be explored and extended through an initial co-operative evaluation. Once the model is developed it could be analyzed by applying a set of property patterns to the model. These property patterns (which are based on usability heuristics) are applied systematically to the model. The application of these properties either yields the fact that the device supports them or produces a trace where the property is broken. This trace can then be rendered as a scenario – describing a narrative in which the property is broken so that further analysis can be carried out by a usability expert.

The only remaining feature of this iterative process is that the process of enquiry, perhaps using a variety of techniques, may lead to additional properties perhaps relating to the experience of the users of the device that can be formulated and expressed using a tool to aid the generation of properties [12].

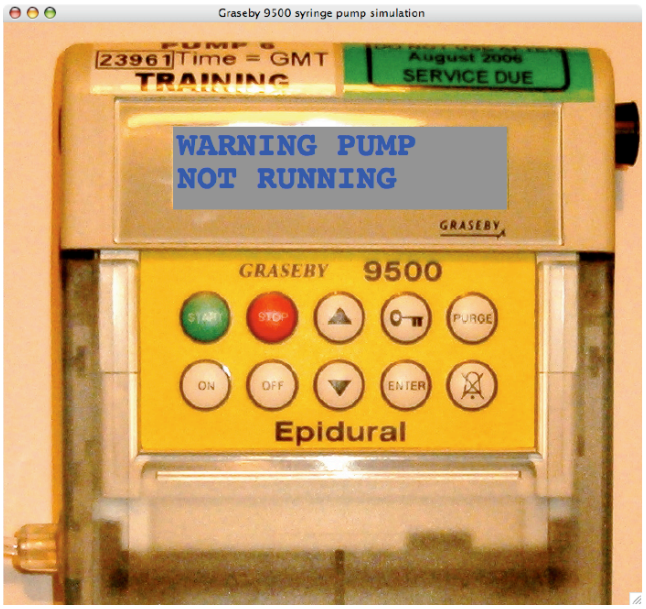

Figure 2. Graseby 9500 (from [16])

4. THE EXAMPLE

The central analysis of the paper concerns the Graseby 9500 syringe pump (see Figure 2). The device is set up by a nurse or anesthetist and injects drugs into patients over a period of hours or even days. It can be used on demand by a patient for pain management. Controlling these devices can involve detailed patient models (for example inputting the weight of the patient). The Graseby 9500 can be worn or carried around by the patient. It delivers calibrated doses of drugs on demand within parameters set up by the nurse. The value of this example is that, in addition to being a safety critical application, it has already been analyzed using a comparable technique in [21]. These two analyses can therefore be compared.

5. THE IVY WORKBENCH

The process described in Section 3 is supported by the IVY tool (see Figure 3). The tool has six components that support these activities (a seventh is mentioned as a desirable component).

(1) A model editor is designed to help the designer to create a model. The model enables the creation of MAL interactor models.

(2) A reverse engineering component takes the code of a prototype of the device and generates a model from it.

(3) A model animator takes a model and produces a prototype from it.

(4) A property editor eases the expression of appropriate usability related properties. It generates the logical formulae that might be verified by the model checker.

(5) A compiler (i2smv) translates these models into the model checker's input language.

(6) A trace visualizer / analyzer aids the analysis of traces.

(7) A scenario generator creates a scenario from a trace.

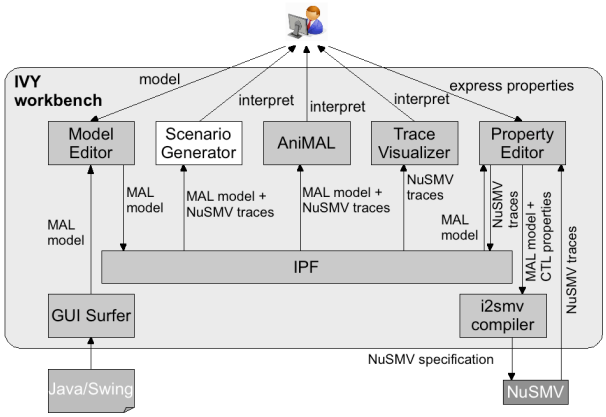

Figure 3. IVY Architecture

Table 1. setting up the effect on states

stop	Infusing	On
	Bolus	Infusion_suspended
	Infusion_suspended	On

All but one of these elements have been completed and are at a prototype stage, while one (the scenario generator) remains to be completed. While many of these features have been explored elsewhere in different contexts the contribution is that these tools are integrated into a single system based around a model checking approach.

The IVY workbench's plugin framework (IPF) is at the core of the architecture, and acts as a broker, providing all plugins with the ability to store and query interactor information. Two basic types of information can be stored: the MAL models, and the traces resulting from the verification step. Plugins are provided with an API that enables them to both store and query that information. Typical examples of the information that can be queried are the attributes and actions of an interactor, or the attribute values of a given state in a trace.

The next sections show how the proposed architecture supports the analysis of the Graseby 9500 syringe pump user interface.

5.1 The model editor

Models are developed in the MAL interactor language [4,5,9]. A MAL interactor is defined by: a set of typed attributes that define the interactor's state; a mapping of the interactor's state to some presentation medium; a set of actions that define operations on the interactor's state; a set of axioms written in MAL (Modal-Action Logic) that define the semantics of the actions in terms of their effect on interactor's state.

In the current case, modeling the system is trivial since it is based on the model used in [21]. This model is available as a state transition table, and can be obtained from: http://www.cs.swan.ac.uk/~csharold/presson/. The table captures how the system state changes in response to different actions. Table 1 presents three rows for illustrative purposes. What it shows is that pressing the Stop button when the system is in either

Infusing or Infusing suspended mode makes it go into the On state, while pressing Stop when in Bolus mode makes it go into Infusion suspended mode. Hence, the model mainly captures the moding of the system.

To express the model in MAL it is necessary to define an attribute that captures the state and to encode the state transitions as MAL axioms.

This state attribute is of type GPstate:

```
interactor main
  attributes
  [vis] state: GPstate
```

Type GPstate is defined as an enumerated type, with one value for each possible system state. There are 54 possible values in GPstate. Table 1 illustrates four of them: Infusing, On, Bolus, and Infusion_suspended.

The actions of the system must be defined. The state transition table identifies a total of 11 actions, 2 of which are internal to the system. This gives rise to the following action definition in the MAL model:

```
  actions
    [vis] down, enter key, off, purge, start, stop, on, up
    lock, timeout
```

The first nine actions are user actions and are thus marked with the [vis] modality. The two additional actions (lock and timeout) are internal actions.

Finally, the behavior of the device must be defined. For each row in the table, a modal axiom is defined expressing the state change it describes. The axioms relating to table 1 are:

```
state=Infusing -> [stop] state'=On
state=Bolus -> [stop] state'=Infusion_suspended
state=Infusion_suspended -> [stop] state'=On
```

Each axiom defines the effect (expression to the right of the [ ] operator) of an action (identified in the [ ] operator), when the condition to the left of the [ ] is true. Hence if the state of the device is Infusing and the stop button is pressed then the new value of state will be On.

In addition to modal axioms the notation allows the representation of invariants – for example that an "on-off" light will always be on when the device is in the On state. It also allows the representation of deontic axioms defining under which conditions an action is permitted or obligatory.

The transition table in [21] defines all the behavior of the system and therefore provides the basis for the MAL axioms used in this analysis. Besides the modal axioms described above, there is the need to state that only those behaviors described in the transition table are possible. For each action a permission axiom is defined restricting behavior to those conditions for which a row is defined. For the stop action case for example:

```
per(stop) -> state in {Infusing, Bolus, Infusion_suspended}
```

Hence, the stop action only causes an effect when the state is one of "Infusing", "Bolus" or "Infusion_suspended".

In the current case a specific given model has been used. In the general case a developer only has to identify what the key attributes of the state are from the point of view of the interaction and then to identify how these attributes are affected by the possible actions. A model is constructed by composing interactors in a hierarchical form. Hence a model can always be represented by a state machine, where the states are defined by the attribute values and the transitions are labeled by the actions that cause changes to the attributes.

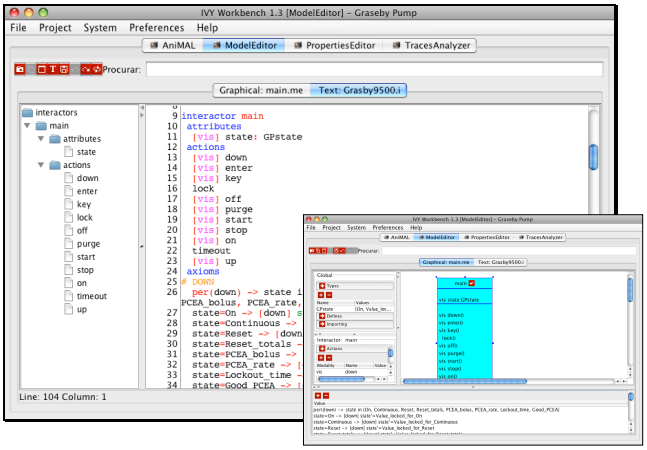

Figure 4. IVY Model Editor

The editor (Figure 4) supports the editing of interactor models in two modes. The first mode allows the developer to visualize and manipulate the overall structure of the model. The model structure is reflected as a hierarchy of objects. The graphical notation used in graphical mode is inspired by UML class diagrams [3]. Interactors are represented as classes, and interactor aggregation and specialization are represented in the usual UML way. UML has the advantage of being a widely known modeling language, hence adopting its notational conventions facilitates comprehension of the model's representation. In graphical mode, besides the graphical representation of the model, a number of inspector panes are provided, enabling editing of different aspects of the model (types, attributes, actions and axioms of the currently selected interactor; aggregations; etc.).

Textual mode, on the contrary, allows direct editing of the text of the model. It provides typical text editing facilities. This mode provides a more effective approach to editing and fine tuning an already existing model. More experienced users of the tool are able to edit the model more quickly as a result, looking for aspects to change directly in the text instead of in the inspector panes of the graphical mode, while still being offered a degree of assistance. Less expert users, on the other hand, can choose the graphical mode, where they are offered more guidance.

5.2 Reverse Engineering Prototypes (GUI Surfer)

This component extracts elements of a model from a Java / Swing program using a slicing function [20]. It isolates the Swing sub-program from the entire Java program. A small set of abstractions are defined for the interactions between user and system: user input, user selection, user action and output. The names of the

attributes in the interactor model are derived from the names of the widget variables in the code. The axioms are inferred firstly by determining the initial state of the code in terms of the attributes that are extracted. An analysis of the potential changes in the user interface, caused by the available user actions is then performed.

The analysis of code is restricted to user interface code. Hence, generated models tend to have a high degree of non-determinism. The IVY editor is used to complete the axioms.

In the current example there was no need to use this component. [21] provides a prototype based on the finite state model.

5.3 The Property Editor

The purpose of the property editor (see Figure 5) is to apply property patterns systematically to the device model. The properties for verification are written in CTL (Computational Tree Logic) [7]. The IVY user can select, from a number of patterns, the property that best suits the analysis needs, and instantiate it with actions and attributes from the model. The correct CTL property for verification is generated by the property editor. A number of patterns are available for application, for example Feedback, Behavioral Consistency, Undo, Reversability, as well as the Dwyer patterns [10]. For more detail about property patterns see [5].

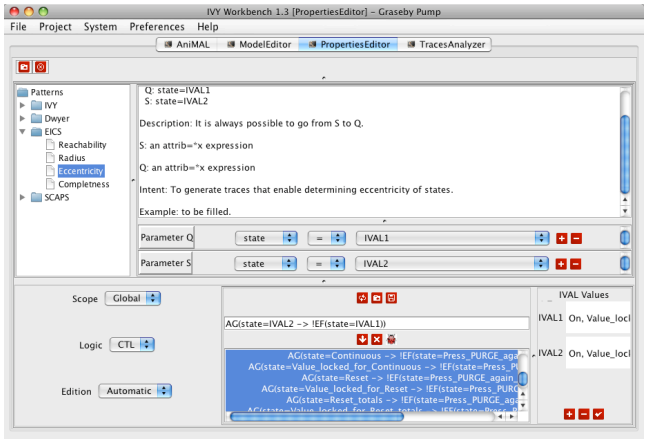

Figure 5. IVY property editor

The system supports quantification of these properties over the state attributes that have been defined in the model. Properties with quantified variables act as meta-formulae that, when instantiated, give rise to a number of concrete valid CTL formulae for verification. This approach provides an increased level of expressiveness. Hence, for example, the visibility pattern can be applied systematically to all actions by writing a single property. Verification results are provided for each concrete instantiation of the property (i.e., in this case, on an action by action basis).

In the next section the existing patterns will be used to support some of the analysis proposed in [21]. In the process some new patterns will also be required.

5.4 Animating Models (AniMAL)

The AniMAL plugin enables user interface prototyping from MAL models (see Figure 6). Currently, prototypes can be animated to present the sequence of events recorded by a fail trace, providing a more textured indication of what went wrong.

AniMAL uses interactor models published by the Model Editor, as well as traces provided by the Traces Analyser. It collects information about all prototype relevant attributes and actions in the model (those with a modality). These are made available as a basis to construct a prototype. It is then possible to drag available elements to the view port to create a visual representation of the interactor attributes and actions as appropriate. Furthermore, automatic prototype generation can be requested, rendering all elements using default components.

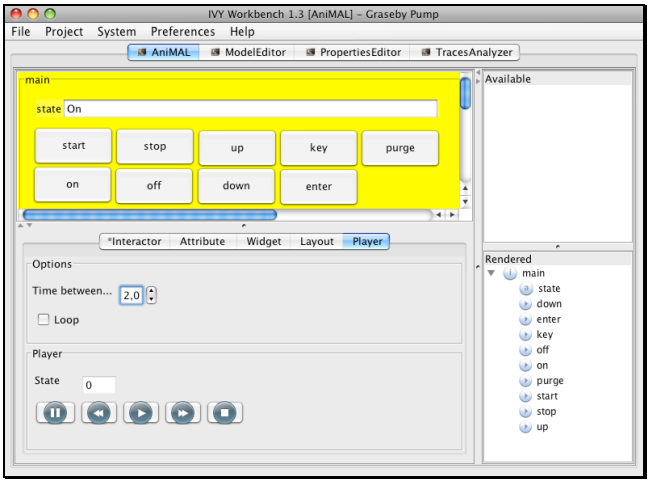

Figure 6. The AniMAL plugin animating the Graseby 9500

Each interactor is represented by a panel, grouping the interactor's attributes and action representations. Interactor composition is represented by the nesting of their panels. Once rendered, elements are placed hierarchically, using the main interactor's panel as the root on the view port.

Attributes should be rendered flexibly. An attribute may need to be rendered with one of several components, to have its values represented more realistically or conveniently (for example, to include an explanatory label next to an input field). There are several data types for storing values (for instance, Boolean attributes require different components than integer attributes). The process of rendering attributes during animation requires synchronization with the data model. The approach used requires each widget to implement a set of interfaces, making it possible to write new widgets and use them to represent attributes without requiring changes to the plugin. For simplicity's sake, actions are currently rendered as toggle buttons.

The prototype is built around the known Hierarchical Model-View-Controller (HMVC) pattern, using the *Tikeswing* framework (http://sourceforge.net/projects/tikeswing (27 February 2009)). Interactor panels are represented by a Model-View-Controller (MVC) set. The *View* aggregates graphical elements, like child interactors' views, as well as attribute and action widgets. The *Controller* handles view events, like mouse drags and keyboard events as well as View-Model synchronization. Finally, a centralized *Model* stores attribute and action values.

Using the HMVC pattern, values are transparently synchronized between model and view, eliminating the need for complex event handling and dependent value updates of user interface components. Therefore, a change to an attribute's value in the model is automatically reflected in the view represented in the corresponding widget.

To adapt the prototypes to different situations, AniMAL enables the user to select different customizations, creating a more realistic user interface. Currently, three levels of customization are possible: changing the layout management algorithm; changing the widget associated with the given attribute; and changing the rendering properties of a given widget.

Each fail trace includes a set of values for each attribute and actions for each state. Animating a fail trace is done by changing the data model to match each state sequentially in the trace. By using the HMVC design pattern, only the data model needs to be changed, while graphical elements are automatically updated, due to View-Model synchronization.

5.5 Verifier

The verification step is performed by the NuSMV [6] model checker. To make the verification possible, MAL interactor models are compiled to the SMV language by the i2smv compiler. When a given property is not verified, SMV provides a behavior of the model (a trace) that demonstrates that the property in question is false. A trace consists of a sequence of states of the model that violates the property under scrutiny. The states can be analyzed with the Trace Visualizer component described next.

This section focuses on verification performed on the Graseby device. As discussed in the introduction, two contrasting analyses were performed. The first type of property is concerned with analyzing features of the interactive behavior of the device to explore situations where usability properties fail. These properties are applied successively over the possible values of the state attributes as quantified in the property template. The second type of property is concerned with the user effort associated with achieving different activities. Here the concern is with graph properties of the device. The discussion that follows shows how the IVY tool can support much of the latter style of analysis. The properties analyzed are developed in the same order as they appear in [21].

5.5.1 Reachability properties

These can be verified using existing IVY property templates. In particular, the **Reachability** pattern:

AG(state=*x* -> EF(state=y))

where *x* and y are variables that range over the possible states of the system.

CTL does not allow variables to be quantified over states but, as discussed above, the IVY tool can be used to instantiate the property as required with all the possible values for *x* and *y* ranging over state. Since there are 54 possible states, this generates 2,916 different properties. The tool quickly checks the full set. All instantiated properties hold true. It can be concluded therefore that the model is strongly connected. If the property had failed, the counter examples would show which states are not reachable from which other states. If reachability of states from one particular state is of interest (as is discussed in [21]) then it is straightforward to instantiate the particular value of *x* and write:

AG(state=on -> EF(state=y))

to check that all states are reachable from this particular state. Alternatively, the same property can be checked using the **Possibility** pattern, while defining On as the initial state of the model:

AG(EF(state=*x*))

The Possibility pattern instantiates 54 properties. These are all quickly proved.

The next analysis performed by [21] is concerned to "determine the set of states that can reach selected states". This can also be achieved using the **Reachability** pattern. For example, checking the subsets of states that reach the Infusing state can be achieved by using:

AG(state=*x* -> EF(state=Infusing))

The required subset is composed of those states for which the verification succeeds. In the present case, since the model is strongly connected, all properties succeed.

5.5.2 Eccentricity

In [21] the notion of eccentricity is used to define the diameter and radius of the graph. This enables the identification of its periphery and center and is considered to be a measure of the user effort required by the interface. Calculating eccentricities can be done using a pattern that has been added to the IVY tool, the **Eccentricity** pattern

AG(state=*x* -> !EF(state=y))

Eccentricity is determined by the longest path between x and y returned by the model checking step. Calculating the eccentricity of the state On can be achieved by:

AG(state=on -> !EF(state=y))

The result of this analysis is to conclude that its eccentricity is 5. This approach works because the model checker returns the shortest path between state On and state y (for every possible y). At the moment, determining the longest path must be done by visual inspection using the trace visualizer plugin. Automating the process will be easy to achieve.

5.5.3 Completeness

Completeness refers to the ability to reach all possible states with one action. It can be verified with the following pattern (which is added to the IVY template battery as **Completeness**):

AG(state=*x* -> EX(state=y))

Instantiating this pattern produces 2,916 more properties. In this case the verification takes longer (over one hour on a Mac Book with an 2.2GHz Intel Core 2 Duo processor and 4GB of memory). It is concluded that the dialog is not complete. The counter-examples identify those pairs of states for which there are no single step transitions. Notice that extending this to consider a multi-step path would amount to simply increasing the number of EX operators in the formula.

5.5.4 Undo

Undo is a standard property analyzed by the previous IVY process. To analyze "undo equivalents" an undo pattern can be used. To test whether stop and start are undo equivalent the "**Undo (by specific action)**" pattern is used. It fails for:

```
AG (state = Infusion_suspended -> AX (action = stop ->
        (EX action = start & AX (action = start -> state =
                                    Infusion_suspended))))
```

That is, when the system is in the infusion suspended state, and stop is pressed, pressing start does not undo the stop effect.

A trace is produced that highlights the problem. The trace analyzer shows that when in Infusion_suspended mode, pressing stop and start puts the system in infusing mode (not infusion *suspended* mode). Considering that start undoes the effect of stop in all other circumstances points to a potential usability problem.

To demonstrate the issue with an end user the AniMAL plugin can be used to create a simulation of the system and run the problem trace (see Figure 6). This will help validate the likelihood of the situation being a real usability concern.

Because undo equivalence need not be symmetric, the pattern is instantiated for the start-stop combination also. This analysis succeeds only when the system is in the On or Infusion_suspended states.

The above analysis makes it possible to verify whether two given actions are undo equivalent. To determine undo equivalent pairs, a new pattern is defined (**Undo Equivalent pairs**):

AG(state=*x* -> AX(Q -> !EX(state=*x* & !R)))

where Q is the action to undo and R a set of actions that are not to be considered. This will produce traces, each identifying one action, not in R, that undoes Q.

Using this pattern, it becomes possible to determine which actions are undo equivalent. However, the pattern "fails" silently (in fact, the verification will succeed) if an action has no undo equivalent. Using the **Undo** pattern quickly determines which actions are undoable and under which conditions. At the same time it is possible to determine the minimal cost of undoing an action using:

AG(state=*x* -> AX(ac1 -> !EF(state=x)).

With this information, it becomes possible to calculate the undo cost of the system as defined in [21].

5.5.5 Other costs

Determining overrun cost is possible, but it must be defined explicitly through the sequence of actions to be analyzed. This can be done using AX operators in a formula similar to the one above.

Reset recovery cost can be calculated by the following pattern (**Reset recovery**):

AG(state=*x* -> AF(state=Q & !EF(state=*x*)))

where Q is the reset state.

Instantiating the pattern, with x ranging over all possible states, generates properties whose verification produces traces identifying minimal paths from each state to Q (for example, off) and back to the original state.

5.5.6 Topological properties

The relation of a number of graph properties, like edge connectivity and minimum cuts, to usability is also explored in [21]. These are topological properties of the graph which are complementary to the behavioral analysis illustrated above. Model checking is not appropriate in that case.

5.6 The results of the verification

The next stage involves analysis of the behavioral traces resulting from the verification step. The visualization component aims to ease recognition, helping to determine what the problem is that is being pointed out by a trace, helping to discover possible solutions to it. In developing the IVY tool four different formats of traces were explored: a tabular representation similar to the existing SMV representation, part of the Cadence Labs tool (www.kenmcmil.com); a Physical States representation showing a graphical representation of the trace states; a Logical States representation which is similar to the physical states representation but the trace states are pre-processed to eliminate artificial states introduced by the compilation process; an Activity Diagram representation which is centered on actions described by Activity Diagrams (UML 2.0 [3]).

5.6.1 Tabular

In this representation (see Figure 7) the information is presented in a table, in the style of Cadence SMV tables [14]. The headings in the columns show state numbers. The beginning of a cycle is shown using an asterisk.

One cell with a colored background indicates that the attribute's value in the current state has changed when compared with the same attribute's value in the previous state. When the value remains from one state to another, the white background is used (in the case of Figure 7, all cells are colored). This idea, adopted from [14], aims to allow rapid recognition of when the interactor's attributes change state.

	1	2*	3	4
main.action	up	stop	lock	enter
state	Infusion_suspended	On	PCEA_bolus	PCEA_rate

Figure 7. Tabular visual representation

5.6.2 Physical States

This representation (see Figure 8) is inspired by state transition diagrams. Each interactor is represented in a column which shows the states that the interactor goes through in the trace. The global state, with all the interactor's variables, is also represented to provide the analyst with an index of the state attributes. This index can be used to see, at each step, the global state formed by the states of each individual interactor. In this global state a green arrow indicates the beginning of a loop, with another arrow indicating its end in the last state.

The interactor's attributes are either shown next to each state or can be toggled off to provide a more compact view. In that case, attributes are not represented in the diagram, but can be consulted by placing the mouse over each state. The fact that information does not appear all the time on the screen (with pop-ups activated) helps the user diminish the amount of information that it is necessary to analyze in order to discover the problem highlighted

by the trace. Actions are shown as labels in the arrows between two consecutive states. These arrows are only shown if a transition exists for the given interactor in that particular state. When the arrow is not shown, this indicates that the state of the particular interactor has not changed even though the system has continued to progress through the transition of other interactors.

While this representation is effective, states are represented at the same level of abstraction as the SMV code. This issue is dealt with in the logical representation.

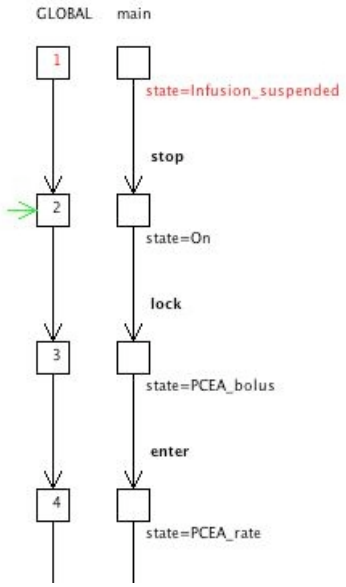

Figure 8. Physical States visual representation

5.6.3 Exploring the traces

To facilitate the analysis of the traces, the visualizer allows the possibility of marking states depending on the criteria defined over the state attributes. In this case the criteria are defined in a panel that is in the main window (see Figure 9). The criteria are defined using relations (=, >, <) between attribute pairs or between attributes and values. To each criterion is associated a color. All the states that verify a given criterion are annotated with this color.

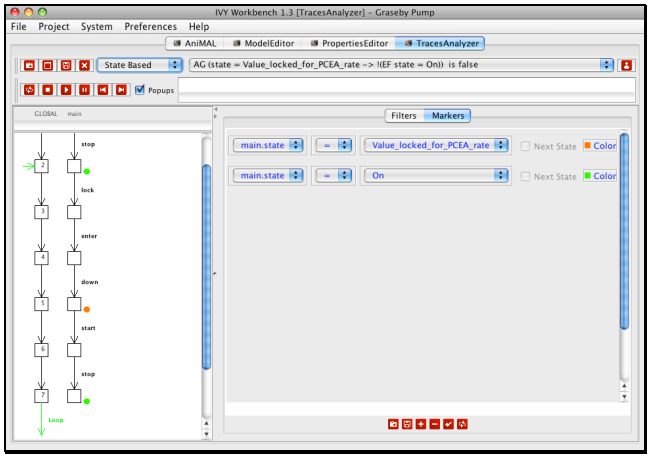

Figure 9. Analysis of the example in the Logical States representation using markers

In the case of comparison between attributes, two half-circles are drawn with the chosen color. Each half-circle is drawn near each of the attributes to indicate that the relation holds. In the case of comparison between attributes and values, filled circles are used in the chosen color. If the pop-up option is enabled it is possible to consult the condition represented by each marker by placing the mouse over each attribute.

6. COMPARING TRACE REPRESENTATIONS

The problem that trace representations aim to solve is to support the developer in finding the problem in the model (if there is a problem in the model) or, alternatively, providing a basis that the developer can use to communicate with the human factors expert. The aim therefore is to ensure that the representation can serve these roles. There is still work to do here, however a preliminary evaluation has been completed to explore the effectiveness of the tabular and physical state representations. This study was based on the model of the Toyota air conditioning system reported in [5]. It is preliminary, designed to be indicative and formative in choosing an appropriate trace representation, both as a basis for analysis by the developer and as a basis for the scenario.

In the study the subjects were Computer Science students at the University of Minho. The study involved two groups, eight in each group. The two groups were presented with the results of two verification attempts. The first attempt addressed an undo related issue (whether pressing a given button twice would leave the device in its original state). The second example, which is more complex, aims to prove that if the Air Conditioning is on, and the environment temperature is higher than the temperature defined by panel, then the environment temperature will reduce until it reaches the panel temperature. In the study the subject's task was to interpret the counter examples as presented by the trace representations and to understand where and why the verification has failed.

The two groups both saw the simpler problem first. Each group was split in two and the two representations were switched between each sub-group so that the first sub-group saw the tabular representation first and vice versa. A co-operative evaluation approach was taken and after the evaluation each subject was asked to complete a questionnaire.

The evaluation proceeded as follows:

1. The model is explained to the subject while at the same time describing how the IVY tool works and in particular how the model checking process works.
2. The property for which the trace is a counterexample is explained to the subject.
3. The subject is asked to offer an opinion about the representation, what is happening in the trace (they have the model in front of them) and whether they can identify the problem that the trace represents.
4. This process is repeated in the case of the other trace, with the different trace representation.
5. The subject is then asked to complete the questionnaire.

The questionnaire was adapted from one based on the USE framework (assessing three dimensions: Usefulness, Satisfaction,

and Ease of use [15]). This questionnaire was adapted to: allow the evaluation of representations instead of user interfaces and to use a comparative item analysis as opposed to a five point Likert scale.

The results are indicative only. As stated above, test subjects were split according to their prior knowledge of model checking. For those without prior knowledge of model checking, the tabular representation was harder to understand on first impact. In the beginning, subjects did not identify the actions and the state transitions. After a short period (minutes), all subjects identified the problem and the reason for it. The Physical States representation appeared to be easier to understand in the sense that subjects understood where the actions and state transitions were.

Two subjects who began with the Tabular representation looked at the table, interpreted it as a database table, therefore did not identify the state transitions, and therefore did not identify the problem. After an explanation these subjects completely understood the problem and identified the reason for it. None of the subjects using the tabular representation identified the loop in the representation.

Users with previous knowledge of model checking seemed to be comfortable with the tool. Both those who started with the tabular representation and those who started with the physical states diagram were able to identify all the interface components. They easily understood which action was being represented in each state transition of the counter-example. They were able to identify where the condition was being violated and its causes. The problem was well identified in both examples (the easier and the harder).

One of the subjects that started with the tabular representation was not capable of interpreting it. He could only identify the problem after further explanation of the interface components.

The results of the questionnaires provided information about preferences. In terms of usefulness, subjects did not show a clear preference for either representation. The group that was familiar with model checking showed a slight preference for the tabular representation whereas the group unfamiliar with model checking appeared to show a preference for the physical states representation. In the case of ease of use there was a clear preference for the state representation by both groups. No preference was shown in terms of ease of learning.

7. CONCLUSIONS

The paper has focused on two issues. It has described the main ingredients associated with supporting an analysis process using the IVY tool. This tool is concerned with modeling interactive devices and the analysis of models using model checking tools. It is concerned with the means of presenting the results of the analysis both in terms of animations or prototypes of scenarios and the understanding of the scenarios themselves. The paper includes a preliminary evaluation of the traces that are output.

The paper raises questions about the nature of usability evaluations, challenges that arise because of the comparison with the results of [21]. Previous work relating to IVY and other research by the authors makes the assumption that an interactive device should satisfy a set of properties and the interesting situations arise when the properties fail, the discussion has therefore revolved around the traces generating scenarios that are of interest from a usability point of view.

[21] explores a different proposition, namely that usability is at least as much about the user's effort in using the system. It describes this effort in terms of graph properties, the length of paths, the valency of nodes and so on, indicating choice and moding within the device.

In practice usability analysis needs to contribute both styles of analysis. The paper aims to make a further contribution, namely that providing a comparable analysis of a design makes it possible more directly to assess the value of two techniques in the analysis of interactive devices.

8. ACKNOWLEDGMENTS

Many people have contributed to the development of the IVY tool. Special thanks are due to Nuno Sousa for working on the majority of plugins, Sandrine Mendes, Vitor Pinheiro and Nuno Guerreiro, for developing AniMAL. Diogo Martins, Rui Freitas and Nuno Job for performing the usability study pilot. Development has been supported by the *Fundação para a Ciência e a Tecnologia* (FCT, Portugal) and the European regional development fund through grant POSC/EIA/56646/2004. Thanks are also due to Harold Thimbleby and his team for providing the challenge in the first place.

9. REFERENCES

[1] Berstel, J., Reghizzi, S.C., Roussel, G. and Pietro, P.S. 2005. A scalable formal method for design and automatic checking of user interfaces. ACM Trans. Softw. Eng. Methodol., 14(2):124–167.

[2] Blandford, A.E., Hyde, J.K., Green, T.R.G. and Connell, I. 2008. Scoping analytical usability evaluation methods: a case study. Human Computer Interaction 23: 3, 278-327.

[3] Booch, G., Rumbaugh, J. and Jacobson, I. 2005. Unified Modeling Language: user guide. Addison Wesley, 2 edition.

[4] Campos, J.C. and Harrison, M.D. 2001. Model checking interactor specifications. Automated Software Engineering, 8:275-310.

[5] Campos, J.C. and Harrison, M.D. 2008. Systematic analysis of control panel interfaces using formal tools. In N. Graham and P. Palanque, editors, Interactive systems: Design, Specification and Verification, DSVIS'08, volume 5136 of Springer Lecture Notes in Computer Science, pages 72-85. Springer-Verlag.

[6] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. 2002. NuSMV 2: An Open Source Tool for Symbolic Model Checking. In Larsen, K. G. and Brinksma, E., editors, Computer-Aided Verification (CAV'02), volume 2404 of Lecture Notes in Computer Science. Springer-Verlag.

[7] Clarke, E.M., Emerson, E.A. and Sistla, A.P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. Transactions on Programming Languages and Systems, 8(2):244-263.

[8] Cooper, S.E., Ramey-Smith, A.M. & Wreathall, J., A. 1996. Technique for Human Error Analysis (ATHEANA). US Nuclear Regulatory Commission

[9] Duke, D.J. and Harrison, M.D. 1993. Abstract interaction objects. Computer Graphics Forum, 12(3):25-36.

[10] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. 1997. Patterns in property specifications for finite-state verification. In Garlan, D. and Kramer, J., editors, *21st* International Conference on Software Engineering, Los Angeles, California, 411-420.

[11] Gray, W. and Salzman, M. 1998. M. Damaged merchandise? a review of experiments that compare usability evaluation methods. Human Computer Interaction, 13(3):203.261.

[12] Harrison, M.D., Campos, J.C., Doherty, G. and Loer, K. 2008. Connecting rigorous system analysis to experience centred design. In Effie Lai-Chong Law, Ebba Thora Hvannberg, Gilbert Cockton (eds) Maturing Usability: Quality in Software, Interaction and Value. Springer Human Computer Interaction Series. pp. 56-74.

[13] Hollnagel, E. 1998. Cognitive Reliability and Error Analysis Method (CREAM). Elsevier.

[14] Loer, K and Harrison, M.D. 2006. An integrated framework for the analysis of dependable interactive systems (IFADIS): its tool support and evaluation. Automated Software Engineering, 13(4):469-496.

[15] Lund, A.M. 2001. Measuring usability with the USE questionnaire. The Usability SIG Newsletter. (http://www.stcsig.org/usability/newsletter/0110_measuring_with_use.html as at 13th February 2009).

[16] Navarre, D., Palanque, P. and Bastide, R. 2003. A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with computers, Elsevier, Vol. 15/3, pp 309-328.

[17] Nielsen J. 1992. Finding usability problems through heuristic evaluation. In: Proc. of ACM CHI'92 Conference on Human Factors in Computing Systems. New York. ACM 249-256.

[18] Paternò F., Santoro C., Mäntyjärvi J., Mori G. and Sansone S. 2008. Authoring Pervasive MultiModal User Interfaces International Journal of Web Engineering and Technology, Inderscience Publishers, pp.235-261.

[19] Polson, P.G., Lewis, C., Rieman, J. and Wharton, C. 1992. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces, International Journal of Man-Machine Studies, Volume 36(5).

[20] Silva, J.C., Campos, J. C. and Saraiva J. 2007. Combining Formal Methods and Functional Strategies Regarding the Reverse Engineering of Interactive Applications. In Doherty G., Blandford A. (Eds.) Interactive Systems: Design, Specification and Verification, volume 4323 of Lecture Notes in Computer Science, pages 137-150. Springer-Verlag.

[21] Thimbleby, H.W. and Gow, J. 2008. Applying Graph Theory to Interaction Design. In Gulliksen, J.; Harning, M.B.; Palanque, P.; Veer, G.C. van der; Wesson, J. (Eds.) Engineering Interactive Systems Springer Lecture Notes in Computer Science. Vol. 4940. 501-519.