Equivalence Checking for Comparing User Interfaces

Raquel Oliveira **Sophie Dupuy-Chessa** **Gaëlle Calvary**

Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
CNRS, LIG, F-38000 Grenoble, France
Emails: FirstName.LastName@imag.fr

ABSTRACT

Plastic User Interfaces (UIs) have the capacity to adapt to changes in their context of use while preserving usability. This exposes users to different versions of UIs that can diverge from each other at several levels, which may cause loss of consistency. This raises the question of similarity between UIs. This paper proposes an approach to comparing UIs by measuring to what extent UIs have the same interaction capabilities and appearance. We use the *equivalence checking* formal method. The approach verifies whether two UI models are equivalent or not. When they are not equivalent, the UI divergences are listed, thus providing the possibility of leaving them out of the analysis. In this case, the two UIs are said equivalent *modulo* such divergences. Furthermore, the approach shows that one UI can contain at least all interaction capabilities of another. We apply the approach to a case study in the nuclear power plant domain in which several UI versions are analyzed, and the equivalence and inclusion relations are demonstrated.

Author Keywords

User interface; comparison; critical systems; formal verification; equivalence checking; bisimulation.

ACM Classification Keywords

D.2.4. Software/Program Verification: Formal method; H.5.2. User Interfaces: Evaluation/methodology; I.6.4. Model Validation and Analysis

INTRODUCTION

The variety of interaction devices has increased over the past years. These devices range from ultra-small devices, such as smartwatches, to very large devices, such as wall-sized touch screens. UIs are expected to cope with this variety. Thevenin and Coutaz [19] introduced the concept of *Plasticity* as the capacity of a UI to withstand variations of its context of use (*platform*, *user*, *environment*) while preserving usability.

Plasticity provides users with different versions of a UI. Depending on the transformations, these versions may diverge in several directions. In this context, one may be interested in knowing if all UI properties (e.g., UI interaction capabilities and appearance) are preserved once the UI is adapted. If it is not the case, which properties are discarded/added.

Our contribution is a technique to automatically compare plastic UIs. The main motivation is to verify to which extent plastic UIs are similar, meaning that at least a minimum set of interactions are preserved after the adaptation. More precisely, this technique covers two aspects of UIs: interaction capabilities and appearance. We provide a means to representing both interaction capabilities and appearance in one single model, which is used afterwards for comparison.

We cover four cases when comparing UI models: equivalent, equivalent modulo some functionalities, non equivalent at all and when one UI contains at least all functionalities of another one (inclusion). The approach is applicable to any plastic UI, but it is more legitimate in critical systems, due to the strong implications of bad UIs in such systems.

The reminder of the paper starts by explaining a nuclear power plant case study. Then several versions of a plastic UI are illustrated, and the comparison criteria are explained. An approach to compare UIs is described, followed by definitions to support UI comparison. We illustrate how the approach is applied to the case study, how it is validated and some discussions. Finally, the related work is presented, concluding with current results and perspectives.

CASE STUDY

This case study relies on a nuclear power plant system prototype that provides an overview of the plant state (Fig. 1, in French). It notifies the control room operator about all unexpected events in the plant. The main UI contains four zones:

1. The top part displays six tabs for selecting the plant status, which can range from RP (working at full capacity) to RCD (completely stopped).

2. The *Default Signals* (“Signaux de défaut”) zone synthesizes signals triggered in reactor functions, according to unexpected events occurred in the parameters.

3. At the bottom (“Paramètres”), various reactor parameters are displayed (e.g. pressure), each one represented by a widget containing: the parameter name, its current value, a curve with the value evolution over time, a min/max value bounds, which sensor monitors the parameter and its measurement unit. If unexpected events occur in some parameter, the same is highlighted (e.g. a stronger frame around it), and a signal is triggered in the zone two of the UI.

Paste the appropriate copyright statement here. ACM now supports three different copyright statements:
• ACM copyright: ACM holds the copyright on the work. This is the historical approach.
• License: The author(s) retain copyright, but ACM receives an exclusive publication license.
• Open Access: The author(s) wish to pay for the work to be open access. The additional fee must be paid to ACM.
This text field is large enough to hold the appropriate release statement assuming it is single spaced.
Every submission will be assigned their own unique DOI string to be included here.

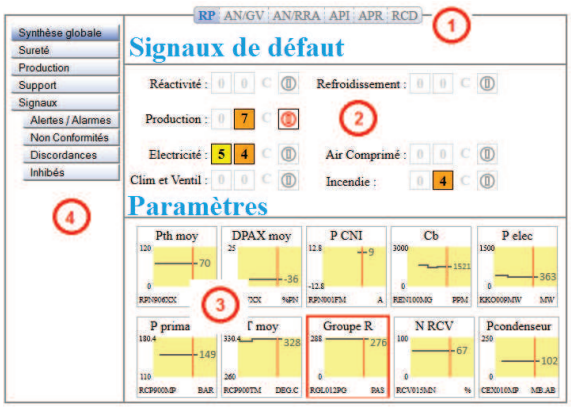


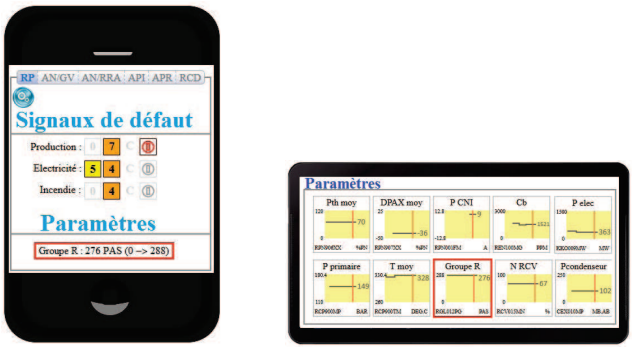
Figure 1: Prototype of a nuclear power plant control room system. Main UI of the system - PC Version

4. On the left, operators access other UIs by a menu. Some of these UIs (covered by this study) have the same layout of this main UI, varying the parameters and signals. Other UIs (not covered by this study) have different purposes.

HCI CONTEXT

Two adaptation means among the seven identified by Vanderdonckt, Calvary *et al.* [21] have been explored for this case study: re-molding and redistribution. UI *re-molding* denotes any UI reconfiguration that is perceivable by the user and that results from transformations in the UI, while *redistribution* denotes the re-allocation of the UI components to different interaction devices. These *adaptation means* do not mutually exclude one another. A redistribution can be followed by a re-molding on the target device, for instance.

Fig. 2a illustrates an example of re-molding: the control room UI is adapted to the target platform (a Smartphone). This UI makes operators mobile, which is useful when an unexpected event occurs in the plant. While on the PC version (Fig. 1) all reactor signals and parameters are always displayed, on the Smartphone the display is limited to those currently affected by a failure. Besides, the widget representing reactor parameters is re-molded to fit on the size-reduced screen of



(a) Smartphone UI (b) Tablet UI

Figure 2: UI Platform adaptation

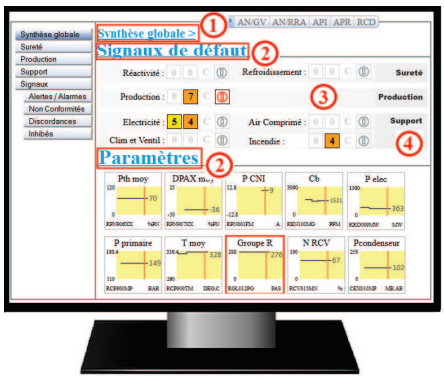


Figure 3: UI in Training Mode

a Smartphone. Furthermore, while on the PC the menu is always visible (in the zone four), on the Smartphone it is accessible by a circled button on the top-left corner.

Fig. 3 and 4 illustrate another example of re-molding, in which the UI is adapted to the target user. This adaptation considers two outermost cases in operators training process: Training mode (Fig. 3), for operators learning how to use the system, and Expert mode (Fig. 4). Fig. 1 is an intermediate mode. The following elements are added in Training mode: 1) at the top, a breadcrumb trail helps navigation; 2) UI zones 2 and 3 are entitled; 3) Non-failure signal symbols have a disabled appearance (e.g. the four symbols beside the "Air Comprimé" function in the Default Signals zone); and 4) Reactor functions are line-grouped according to their systems: Safety ("Sureté"), Production, or Support. In Expert mode, all this guidance is removed.

A Tablet version (Fig. 2b) of the UI illustrates *redistribution*. The UI is re-distributed on a tablet, but only part of the UI is migrated (i.e. the "Parameters" zone), the other part is displayed on other devices, such as kiosks.

Re-molding and redistribution transform a UI into various versions. We propose an approach to show to what extent these UIs differ. This work covers two UI aspects: interaction capabilities and appearance.

Figure 4: UI in Expert Mode

UI *interaction capabilities* concerns the ways users can interact with the UI (and, reversely, how the UI reacts to this interaction). When comparing two UIs, we want to know whether users can perform the same actions in both of them, and whether the UIs react in the same way or not. In this point of view, we are interested in *what* the user can do (e.g., "select menu option 1") , and in *what* the UI does in reaction (e.g., "display main UI"). It concerns neither *how* such interaction capabilities are provided (e.g., which widget is used to display the menu) nor *how* the UI displays the outputs. This relates to UI appearance.

UI *appearance* concerns the elements present in the UI (where they are presented, in which color, etc.). For instance, we may want to know which symbol represents the absence of unexpected events in the reactor.

OVERVIEW OF THE APPROACH

The first step (1) of our approach (Fig. 5) consists in creating a formal model of the UIs. A formal model is an abstraction of the system. Abstractions done in this phase should not penalize the analysis, meaning that the formal model should reflect as much as possible the real system.

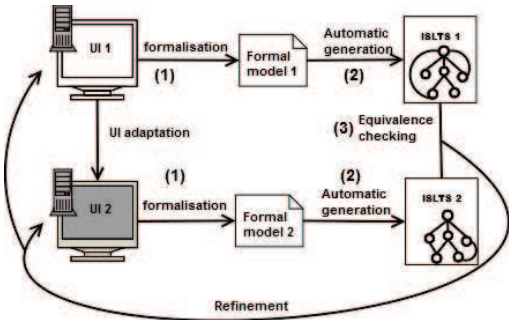


Figure 5: Comparison of UIs

The formal model is used to automatically generate (2) an ISLTS (Interactive System LTS). We derived ISLTS from LTS (Labelled Transition System), a graph representing all the system states, and the actions that trigger state transitions.

The verification of ISLTS equivalence (3) is performed thanks to the *equivalence checking* formal method (Fig. 6). First, we verify whether the two ISLTS (UI models) are equivalent or not. If they are, considering interaction capabilities and appearance, the two UIs are equivalent. Otherwise, we verify if

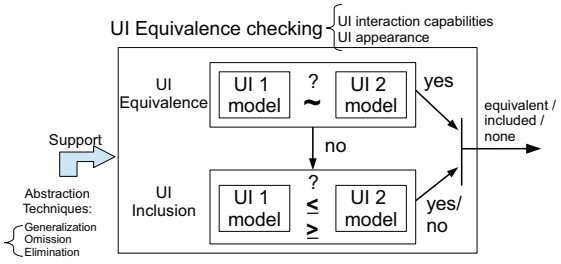


Figure 6: Equivalence checking of UIs

one *includes* the other, which means that one UI contains at least all interaction capabilities (and corresponding appearance) of the other one. The third possibility is that the two UI models are neither equivalent nor included. The analysis is supported by three abstraction techniques, which are explained in the following. The results of the comparison allow a refinement of the formal models and/or the real UIs.

UI COMPARISON

We enhance LTS to model UIs. LTS is a 4-tuple $\langle Q, A, T, q_0 \rangle$ consisting of: a set Q of states; a set A of actions; a transition relation $T \subseteq Q \times A \times Q$ and an initial state $q_0 \in Q$.

Definition 1 (ISLTS)**.** *An* ISLTS *(Interactive System LTS) is a 6-tuple* $\langle Q, C, L, A, T, q_0 \rangle$ *where:*

- Q is a set of states the UI can be in;
- C is a set of UI components;
- L is a set of action names;
- A is a set of actions. They model the system dynamics: actions users can perform in the UI and the UI response to these actions. Each action $a \in A$ has the form $l(c_1, ..., c_m)$, where $l \in L$, $m \geq 0$ and $(\forall i \in [1..m]), c_i \in C$. Intuitively, actions can carry a list of UI components, representing the UI appearance after the action is performed;
- $T \subseteq Q \times A \times Q$ is a transition relation that changes the UI state once an action $a \in A$ is performed. We also use the notation $q \xrightarrow{a} q'$ for $(q, a, q') \in T$;
- $q_0 \in Q$ is the initial state of the UI.

An ISLTS provides a means to representing both UI interaction capabilities and appearance in one single model: they are represented in the set A of actions. Each action represents an interaction capability, and ISLTS enriches LTS actions with data (the set C of components) to represent UI appearance. According to the domain, the set C is composed by subsets detailing the components of the UI.

Figure 7: UI appearance in an ISLTS

Fig. 7 illustrates an example of an action representing the display of reactor parameters, their current value and failure condition. Consider a subset $S = \{normal, fail\}$ of reactor parameter status and a subset $P = \{Pth_moy, GroupeR, ...\}$ of reactor parameter names. Concerning UI appearance, the C set represents how the reactor parameters are displayed in the UI, i.e with their values and failure condition. In this example, $c \in C$ has the form $p(v, s)$, where $p \in P$, $v \in \mathbb{R}$ and $s \in S$, i.e. $C = \{Pth_moy(70, normal), GroupeR(276, fail)\}$. The C set is domain-dependent and can have other formats, not changing the way it is integrated in ISLTS actions, i.e. $a \in A = l(c_1, ..., c_m)$.

In this example, $L = \{ShowParams\}$ and $A = \{ShowParams(Pth_moy(70, normal), GroupeR(276, fail))\}$.

Equivalent user interfaces

Once the UIs are modeled as ISLTS, we can perform UI comparison thanks to equivalence checking formal technique. We introduce now several definitions we derived from formal techniques to apply to HCI. We combine the notion of equivalence with several abstract criteria.

Definition 2 (Equivalent user interfaces). *Given two* ISLTS *M and H, if a specific relation R exists (called a* bisimulation*) between the states of M and H, then M and H are* equivalent *(written $M \sim H$).*

We derived this definition from *bisimulation equivalence* formal definition. The relation R can be defined by several *bisimulation relations* available in the literature, such as *strong bisimulation* [16] and *branching bisimulation* [20]. Which relation to choose depends on the level of details of the model and the verification goals. We use *strong* and *branching* bisimulation relations, due to the strong implications provided by former and to the flexibility provided by the latter.

Strong bisimulation relation is the most restrictive relation. It relates two standard LTS in the following way: two LTS M and H are strongly bisimilar if there exists a relation $R \subseteq Q_m \times Q_h$ (called strong bisimulation) such that:

1. The initial states of M and H are related by R;
2. If $R(m, h)$ and $m \xrightarrow{a} m'$, then there exists a state h' such that $h \xrightarrow{a} h'$ and $R(m', h')$;
3. Conversely, if $R(m, h)$ and $h \xrightarrow{a} h'$, then there exists a state m' such that $m \xrightarrow{a} m'$ and $R(m', h')$.

This formal definition concerns the LTS states and the actions that trigger state transitions. Such concern for LTS actions suits our UI interaction and appearance modeling in the ISLTS actions. Since in an ISLTS every action a has the form $l(c_1, ..., c_m)$, where $l \in L$, $m \geq 0$ and $\forall i \in [1..m], c_i \in C$, when comparing ISLTS actions both UI interaction capabilities and UI appearance are taken into account.

Strong bisimulation is intuitively illustrated in Figure 8. Two systems (represented by ISLTS) are strongly equivalent whenever they can perform the same actions (possibly enriched by UI components) to reach strongly bisimulation equivalent states, i.e. they agree on each step they take.

Diversely, there are cases in which certain actions (together with the UI appearance after the action execution) may be skipped in the analysis. These actions receive a special label

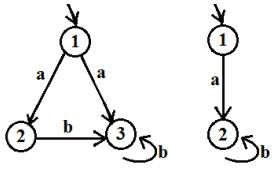


Figure 8: Two strongly equivalent ISLTS

(i.e. τ) in the LTS, and several bisimulation relations exist that deal with τ actions in a special way (*branching bisimulation* is one of the most commonly used). $\tau \in A$ represents an action that is considered irrelevant in the context of the analysis, and can be ignored, although it is still present in the UI model. We call this abstraction technique an *omission*:

Definition 3 (Omission). *Given an* ISLTS $U = \langle Q, C, L, A, T, q_0 \rangle$, Omit(O,U) $= \langle Q, C, L, A \backslash O, T', q_0 \rangle$, *where* $T' = \{(q, a, q') \mid (q, a, q') \in T \text{ and } a \notin O\} \cup \{(q, \tau, q') \mid (q, a, q') \in T \text{ and } a \in O\}$.

Tagging some actions $a \in A$ with a special label (i.e. τ) allows weaker bisimulation relations to bypass such actions when checking the equivalence between models. Since in an ISLTS every action a has the form $l(c_1, ..., c_m)$, once an action a is ignored, a UI interaction capability is intentionally disregarded, together with the UI appearance that results from such action (possibly modeled in the body of the a action).

This abstraction is useful, for instance, when users are provided with a functionality activated in different ways in two UIs: for example, two UIs U_1 and U_2 that have menus with the same options, as illustrated by the sets A_1 and A_2 of actions below. The menu is always visible in U_1 and hidden in U_2, as illustrated by the absence (resp. presence) of the "open_menu" action in the set A_1 of actions (resp. A_2). Once unfolded, the U_2 menu behaves exactly like in U_1. By including "open_menu" in the set O of omitted actions, *omission* permits the menu activation action to be ignored when comparing the UIs, although it is still present in the U_2 model.

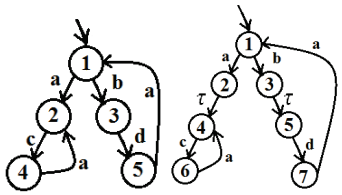
$$A_1 = \{choose_menu_option1, choose_menu_option2\};$$

$$A_2 = \{open_menu, choose_menu_option1, choose_menu_option2\};$$

$$O = \{open_menu\}.$$

When omitted actions (τ) are present in the model, weaker bisimulation relations are more appropriate. *Branching bisimulation equivalence* is one of the most commonly used. It considers sequences of τ-actions. We write $m \Rightarrow m'$ for a path from m to m' having an arbitrary number (≥ 0) of τ-actions. Branching bisimulation relates two standard LTS in the following way: two LTS M and H are branching bisimilar if there exists a relation $R \subseteq Q_m \times Q_h$ (called branching bisimulation) between the states of M and H such that:

1. The initial states of M and H are related by R;
2. If $R(m, h)$ and $m \xrightarrow{a} m'$, then either $a = \tau$ and $R(m', h)$, or there exists a path $h \Rightarrow h' \xrightarrow{a} h''$ such that $R(m, h')$ and $R(m', h'')$;
3. Conversely, if $R(m, h)$ and $h \xrightarrow{a} h'$, then either $a = \tau$ and $R(m, h')$, or there exists a path $m \Rightarrow m' \xrightarrow{a} m''$ such that $R(m', h)$ and $R(m'', h')$.



(a) Without τ-steps (b) With τ-steps

Figure 9: Two branching equivalent ISLTS

Similarly to strong bisimulation equivalence, branching bisimulation also concerns LTS states and actions. The latter compares UI interaction capabilities and appearance as the former: given the form of an action in an ISLTS (i.e. $l(c_1, ..., c_m)$), actions are taken into account with the components present in the UI after the execution of the action.

The essence of branching bisimulation equivalence is illustrated in Fig. 9. These two ISLTS are branching equivalent because for each state, the same actions (preceded by zero or more τ actions) can be triggered.

Regardless the chosen bisimulation relation, our approach permits to reason over UI models at different levels of details. As an illustration, consider a UI fragment of our case study (Fig. 10). This UI fragment displays the number of unexpected events in the "Production" reactor function. The UI on the left represents the absence of unexpected events by a "0" beside "Production", while the UI on the right displays nothing in the same zone. In a more abstract version of these UIs, this information is represented by "default_empty_symbol". We call this kind of abstraction a generalization, and it concerns only UI appearance (not interaction capabilities).

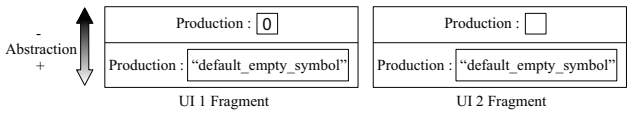


Figure 10: *Generalization* abstraction technique

Definition 4 (Generalization). *Given an* ISLTS $U = \langle Q, C, L, A, T, q_0 \rangle$ *and a set* B *of UI abstract components,* generalization *is a total function* $G \subseteq C \rightarrow B = \langle Q, C, L, A', T', q_0 \rangle$ *that maps every element* c *in* C *to an element* b *in* B*, such that:*

- $A' = \{\tau\} \cup \{l(b_1, ..., b_m) \mid l(c_1, ..., c_m) \in A$ *and* $(\forall i \in [1..m]), G(c_i) = b_i\}$*;*
- $T' = \{(q, \tau, q') \mid (q, \tau, q') \in T\} \cup \{(q, l(b_1, ..., b_m), q') \mid (q, l(c_1, ..., c_m), q') \in T$ *and* $(\forall i \in [1..m]), G(c_i) = b_i\}$.

An example of *generalization* is illustrated below:

$B = \{default_empty_symbol, UI_bottomzone_title\}$;

$C = \{zero, Parameters\}$;

$G = \{(default_empty_symbol, zero),$
$(UI_bottomzone_title, Parameters)\}$.

In the original UI model, the absence of unexpected events in the reactor function is represented by the number "zero". While in a more abstract UI model, the absence of unexpected events in the reactor function is generalized to "default_empty_symbol".

As for the set C of components in the *Definition 1* of ISLTS, the set B of abstract components are also domain-dependent. Besides, the use of regular expressions enables sophisticated transformations in the UI appearance representation. Consider A={ShowParams(Pth_moy(70,normal), GroupeR(276,fail))}, an example of the set A of actions formalized in the *Definition 1*. This action represents the display (in the UI) of reactor parameters with their current value and failure condition. Instead of displaying all reactor parameters, this action could be generalized in a more abstract visualization, where only failed parameters are displayed. In this case, the set B would be defined as $B = \{Failure\ in\ "X"\}$ and regular expressions permit the transformation from A= {ShowParams(Pth_moy(70,normal), GroupeR(276,fail))} into A= {ShowParams(Failure in GroupeR)}.

Non-equivalent user interfaces

From the definition of *equivalent user interfaces*, we propose the definition of non-equivalence:

Definition 5 (Non-equivalent user interfaces). *Two* ISLTS M *and* H *are* non-equivalent *with respect to a relation* R *if the relation* R *between the states of* M *and* H *can not be shown.*

Equivalent user interfaces modulo "X"

There are cases in which certain divergences between two UIs are considered acceptable. For instance, when a navigation aid is present in one UI and absent in another one. Knowing that the UIs present this difference, we may still want to analyze the remaining aspects of the UIs. Equivalence modulo "X" permits this reasoning. We introduce an abstract technique that allows the elimination of elements in the UI model before performing the analysis:

Definition 6 (Elimination). *Given an* ISLTS $U = \langle Q, C, L, A, T, q_0 \rangle$, Eliminate(X,U) $= \langle Q, C, L, A \backslash X, T', q_0 \rangle$, *where* $T' = \{(q, a, q') \mid (q, a, q') \in T$ *and* $a \notin X\}$.

Each action $x \in X$ also has the form $l(c_1, ..., c_m)$, where $l \in L$, $m \geq 0$ and $\forall i \in [1..m], c_i \in C$, so both UI interaction capabilities and appearance are taken into account. Intuitively, $X \subset A$ is a set of actions (and UI appearance) eliminated in U before the comparison analysis. In this case, some UI aspects are left out of the analysis. Contrary to *omission*, in which the elements are still present in the model and are just ignored.

In the example below, extracted from the case study, a breadcrumb trail is present in U_1 and not in U_2. To verify if both UIs are equivalent disregarding this divergence, we eliminate the representation of this functionality in the U_1 model:

$$A_1 = \{select_breadcrumb_trail, choose_plant_state, show_params\};$$

$$A_2 = \{choose_plant_state, show_params\};$$

$$X = \{select_breadcrumb_trail\};$$

$$A_1 \setminus X = \{choose_plant_state, show_params\}.$$

In this case, the two UIs are said equivalent *modulo* the elements of X:

Definition 7 (Equivalent user interfaces modulo "X"). *Given two* ISLTS*, M and H, and a set X such that:*

- $X \subset (A_m \cup A_h)$ *is the set of actions (with the corresponding UI components) simultaneously eliminated in both UI models before performing the analysis,*

if a specific relation R exists (called a bisimulation*) between the states of M and H, then M and H are said equivalent* modulo *"X".*

To show that two UI models are equivalent modulo "X", we consider strong and branching bisimulations. *Elimination* is necessarily used and *generalization/omission* can be used.

Inclusion of user interfaces

Finally, there are cases in which two UIs present a large number of divergences, becoming no longer interesting to eliminate these divergences of the analysis because it will compromise the usefulness of the results.

In this case, the UIs are not equivalent. Even though, they can still relate to each other: one can include the other. For instance, in a control room, operators have at their disposal UIs displayed on PCs to monitor the reactor. Once a UI highlights a failure in a reactor component, mobile operators (provided with a tablet containing only part of the PC-version UI) are charged to perform a maintenance in the proper place and can observe the system reaction on the tablet.

Definition 8 (Inclusion of user interfaces). *Given two* ISLTS *M and H, if a specific relation R exists (called a* simulation*) between the states of M and H, then M and H are* included *one in the other.*

Intuitively, it means that a given user interface U_1 contains *at least* all interaction capabilities (and the appearance) of another user interface U_2.

The same bisimulation relations used to show equivalence between two LTS are also used to show their inclusion (i.e. strong, branching, etc.). We derived our definition of inclusion of UIs from *strong simulation* formal definition, which is achieved when only the conditions 1 and 2 of *strong bisimulation equivalence* hold (*Definition 2*). Two LTS M and H are strongly similar if there exists a relation $R \subseteq Q_m \times Q_h$ (called simulation) between the states of M and H such that:

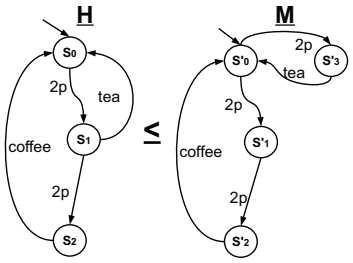


Figure 11: Example of a simulation relation

1. The initial states of M and H are related by R;
2. If $R(m, h)$ and $m \xrightarrow{a} m'$, then there exists a state h' such that $h \xrightarrow{a} h'$ and $R(m', h')$.

Similarly to bisimulation equivalence, *simulation* also concerns LTS states and actions. The latter compares UI interaction capabilities and appearance as the former: given the form of an action in an ISLTS (i.e. $l(c_1, ..., c_m)$), actions are taken into account with the components $(c_1, ..., c_m)$ present in the UI after the execution of the action.

Fig. 11 illustrates a simulation between two ISLTS. $H \leq M$ intuitively means that M can do everything that H can do. M simulates (or includes) H. Generalization and omission abstractions can also be used to show that one UI model includes another one.

DISCUSSION

The abstraction techniques introduced in this paper support UI model comparison. The principle is to first create abstract models of the UIs, used afterwards to perform equivalence checking. Fig. 12 compares the abstraction techniques applied to a UI fragment that considers only appearance. Concerning the level of abstraction, *generalization* technique is the one that abstracts the least, by mapping components into generic representations. *Omission* abstracts more, by obfuscating aspects in the model, and *elimination* is the most significant abstraction, that eliminates UI aspects of the model.

The strongest equivalence relation two UI models can have is when, with none of these abstractions, they are equivalent. This is achieved only when two UIs are almost identical, which is possible, but rare.

In practice, since plastic UIs have to cope with several changes in the context of use, a number of divergences is

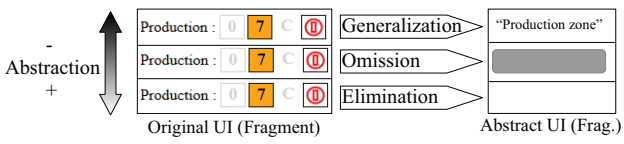


Figure 12: Summary of the abstraction techniques

present within the UI versions. The challenge is to verify equivalence between the UI models in spite of these divergences. Abstraction techniques provide a means to doing that, and weaker equivalence relations between the models can be shown. The more abstractions are used within the models, the weaker the equivalence between the models becomes.

Once the appearance of two UIs diverges, *generalization* is the first technique to be considered. By generalizing the representation of UI components in the models, this technique permits to show a weaker equivalence between models.

Omission and *elimination* are more likely to be used when the UI interaction capabilities diverge. First, by omitting in the UI models interaction capabilities that are punctual, like opening a menu, or opening a combobox, and that do not have a considerable impact in the UIs functionalities. Diversely, elimination is recommended for complex interaction capabilities. In particular, UI functionalities that depend on the system core, that load information in the UI or that enable other UI functionalities. Once such interaction capabilities are present in one UI and absent in the other, elimination provides a means to verifying a weaker equivalence between the two UI models, disregarding (modulo) that.

However, abstraction techniques should be carefully used. The abstraction should never exceed a threshold (manually identified) over which the analysis will no longer be interesting. One should keep in mind that things that are abstracted away are left out of analysis, and interesting situation may be overlooked when the system models become a black box.

APPROACH IN ACTION

This section illustrates an application of the approach to the case study. We show two equivalent UIs, two equivalent UIs modulo one functionality and two non-equivalent UIs that are, even though, included one in the other.

A) Equivalent user interfaces

In order to demonstrate equivalence between two UIs, consider a UI adaptation according to the platform, to which remolding was applied: PC (Fig. 1) and Smartphone (Fig. 2a).

Regarding the UI interaction capabilities, operators have two ways to interact with the UIs: by selecting the plant state and by accessing other UIs using the menu. Users can select the plant status in the same way on both UIs. This was reflected in the ISLTS of both UI formal models by identical states, actions and transitions. The menu, though, is made available in distinct ways: on the PC version the menu is always visible and on the Smartphone it is accessible by a button in the UI top-left corner. Due to these differences in the UIs, the corresponding ISLTS are different. This is illustrated in Fig. 13, with the ISLTS fragments representing part of the menu. Each transition of these ISLTS fragments represents the action of choosing the corresponding menu and sub-menu options.

In this case, "open menu" is an example of τ action: it is a user action that does not have an impact on the available menu options: they are always the same. We used *omission* abstraction to ignore the "open menu" action in the analysis, as if the menu was always visible on the Smartphone UI.

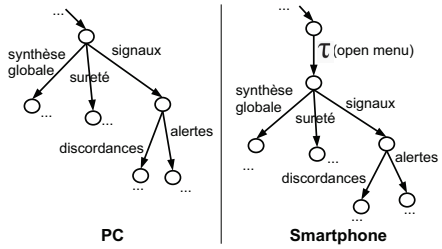


Figure 13: ISLTS fragments of PC and Smartphone UIs

Concerning the UI appearance, both signals and parameters are displayed in the same zones. Due to the lack of space, for this analysis we will deliberately neglect the re-molding in the parameter widgets. We will focus on the way the two UIs display failures: on the Smartphone only the reactor parameters and signals with some failure are displayed, while on the PC all items are always displayed, even non-failure ones. Fig. 14 illustrates such differences in an ISLTS fragment. Both frames on top of the figure represent at a given moment the display of reactor parameters in the UI. While on the PC ISLTS this transition is labeled with an action containing the whole list of reactor parameters, the Smartphone ISLTS contains only the problematic parameter (i.e. "Groupe R"). In this case, we use *generalization* abstraction. Actions containing detailed information are generalized in less detailed actions (i.e. the bottom frames in Fig. 14, with the "Failure in *x*" renamed action).

Using *generalization* and *omission* abstractions, together with branching bisimulation equivalence relation, the PC UI model and the Smartphone UI model were said equivalent.

B) Equivalent user interfaces modulo breadcrumb trail

We demonstrate now two equivalent UIs modulo a particular functionality. Consider a UI adaptation according to the user expertise, to which re-molding was applied: Training Mode (Fig. 3) and Expert Mode (Fig. 4).

Regarding the appearance, there are several differences between the two UIs (detailed in the Section *HCI Context*). We use *generalization* to represent differences 2, 3 and 4 (Fig. 3). In Fig. 15 we illustrate the difference 3: in Training Mode (i.e. the top-left frame), once a given signal is in non-failure status (e.g., "Reactivité"), the corresponding symbols are dis-

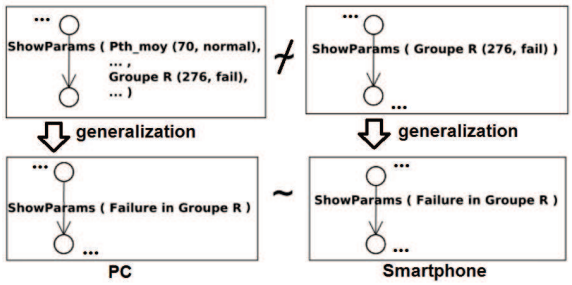


Figure 14: *Generalization* in an ISLTS - case 1

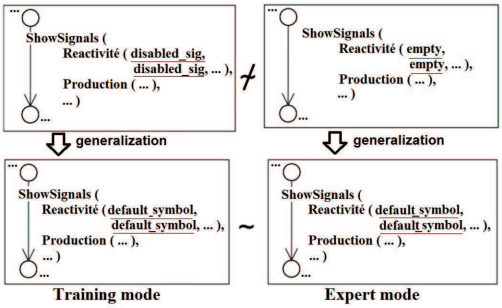


Figure 15: *Generalization* in an ISLTS - case 2

played with a disabled appearance (i.e. "disabled_sig"); in Expert Mode (i.e. the top-right frame), no symbols are displayed beside the signal (i.e. "empty"). Using *generalization* abstraction, in both ISLTS these actions were generalized into "default_symbol" label.

Concerning the UIs interaction capabilities, the Training-mode UI contains one additional navigation aid: a breadcrumb trail (i.e. the difference 1 in Fig. 3). We set the equivalence checking to be done disregarding this feature. We use *elimination* abstraction (Fig. 16) to search (in the ISLTS) actions corresponding to the breadcrumb trail (i.e. the pattern *bct_*). Once a match occurred, all the successor states (and transitions) were eliminated in cascade from the ISLTS.

Using *generalization* abstraction (for the items n. 2, 3, and 4 of Fig. 3) and *elimination* abstraction (for the item n.1), together with strong bisimulation equivalence relation, the Training and Expert UI models were said equivalent modulo the breadcrumb trail. The precise identification of UI divergences is a key contribution of this work.

C) Non-equivalent user interfaces and UI inclusion

We demonstrate now two non-equivalent UIs. Consider a UI adaptation according to the target platform, to which redistribution was applied: PC (Fig. 1) and Tablet version (Fig. 2b).

Regarding UI interaction capabilities, the functionalities that permit user interactions are available only on the PC version (i.e. the menu and the plant status selection). With respect to their appearance, the UIs also differ from each other: the Tablet version does not contain the plant status, the reactor signals and the menu zones.

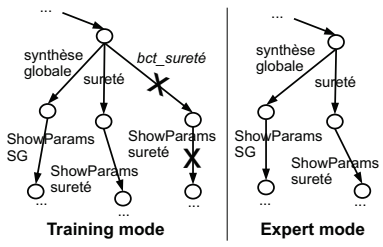


Figure 16: *Elimination* in an ISLTS

The divergences of these two UIs are too large to consider the use of *elimination* abstraction. Indeed, the Tablet-version UI is equivalent to the PC version modulo ["plant-status-related actions", "signals-related action" and "menu-related actions"]. As said in the Section *Discussion*, if we abstract all these actions away, many aspects will be overlooked.

In this case, we applied no abstraction techniques, and the two UI models were shown non-equivalent, because the user can perform several actions on the PC version which are not available on the Tablet version.

Even though, we showed that the PC version contains at least all functionalities (regarding interaction capabilities, and appearance) of the Tablet version. The PC-version UI model included the Tablet-version UI model (i.e. $Tablet_model \leq PC_model$).

VALIDATION

A LNT [5] formal model was manually written for the five contexts of use (Table 1). Such manual specification adds a certain overhead to the process, which is lightened by the fact that the formal languages we use were designed to facilitate formal modeling. This comes mainly from the fact that LNT is an imperative formal language, which easies learning for programmers used to classic programming languages.

Hand-written modeling adds subjectivity to the formal model. To avoid this, the formal models were validated with an expert in the nuclear power plant domain. Such manual modeling requires a considerable knowledge in the UI system, necessary for the analysis of the diagnosis provided by the verification.

Table 1: Summary of the formal models

Context of use	# loc	# states	# transitions
PC	2462	33,053,947	189,539,691
Smartphone	2558	41,944,680	208,554,613
Tablet	1686	4438	5547
Training Mode	2579	160,681,601	946,293,368
Expert Mode	2410	16,678,151	76,202,201

The case study shows that the approach scales well. It was initially designed for one context of use (i.e. PC), later extended to five contexts of use. Each formal model contains three UIs. Each UI model describes about 20 curves and symbols (UI appearance) and 14 user interactions (UI interaction capabilities), generating significantly large ISLTS for the analysis in a reasonable time (maximum 3h). The formal models included also part of the system core, allowing the simulation of several reactor parameters failures. CADP[1] [11] toolbox was used to support the formal verification process, due to the large number of verification tools it provides, specially for performing equivalence checking. Table 1 summarizes the number of lines of the LNT specifications and the ISLTS size. For larger case studies, CADP provides means (e.g. compositional verification) to handle state space explosion, a concern all model checkers have to handle. Model checkers address it by various manners, but human intuition is always needed in the process.

[1] http://cadp.inria.fr/

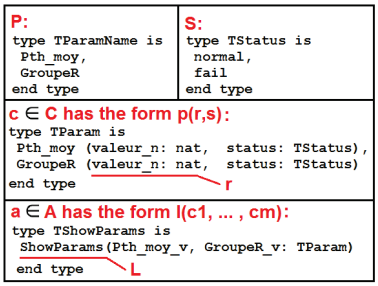


Figure 17: ISLTS in LNT code

Fig. 17 illustrates how the C, L and A ISLTS sets are coded in LNT. Types are defined, and data of such types are exchanged in *ports* by *rendez-vous*, resulting in the ISLTS of Fig. 7.

The abstract criteria were implemented using SVL[2] (Script Verification Language) [10]. While LNT was chosen mainly for its capacity to facilitate modeling, the choice of SVL considerably strengths the approach. SVL offers means to describing operations over LTS, which can hardly be done by hand in large LTS. *Generalization* abstraction was implemented using the "rename" SVL operator, *omission* was implemented using the "hide" operator and *elimination* using the "cut" operator, all together with regular expressions.

SVL scripts implement the three cases described in the Section *Approach in action*. Such scripts transform the ISLTS by means of abstractions, before performing the equivalence verification. Such isolation provided by the SVL scripts spares the original formal models from the abstractions. Fig. 18 illustrates an example of *generalization* in SVL, representing the example illustrated in Fig. 14. Given the "LTS_PC", this script renames all transitions labeled with "SHOW_PARAMS_[anyUI] ([anyParam], FAIL)" into "SHOW_PARAMS (FAILURE in [paramName])", generating a new LTS with the renamed transitions.

```
"newLTS_PC.bcg" = generation of
  total rename "SHOW_PARAMS !UI_.* (.*(\(.*\), FAIL).*)" ->
               "SHOW_PARAMS (FAILURE in \1)"
  in "LTS_PC.bcg"
```

Figure 18: Example of a SVL script

Table 2 illustrates the summary of the comparisons, where O indicates the number of omissions done, G the number of generalizations and E the number of eliminations. The comparison of the ISLTS was done using BCG_CMP[3] and BISIMULATOR[4] tools of CADP.

RELATED WORK

Existing approaches deal with UI comparison in different ways. Some of them compare UIs by user experiments [8, 9], others by classical testing [1, 12, 2] and others are supported by formal methods [3, 7, 13, 6, 17, 18, 14].

[2]http://cadp.inria.fr/man/svl.html
[3]http://cadp.inria.fr/man/bcg_cmp.html
[4]http://cadp.inria.fr/man/bisimulator.html

Table 2: Summary of the comparisons

Models	# O	# G	# E	Result	Comp. time
PC x Smartphone	1	22	0	Equivalent	7min
Training x Expert	0	6	1	Equiv\breadcrumb	19min
PC x Tablet	0	0	0	Tablet included in PC	4s

In [8] several factors to conduct more comprehensive user evaluations of *adaptive* UIs were defined . In [9], user experiments were actually performed with four versions of a UI, in order to measure user satisfaction and preferences. In spite of interesting feedback one can obtain from user experiments, they are very sensitive to the sample of users, and selecting representative users is always a key issue.

In [1] the GUIDIFF toll performs *regression testing* of different versions of a UI, providing a list of their detected deviations. In [12] *Capture-and-Replay* technique was used to perform regression testing of UIs. This technique allows one to re-execute (*replay* part) test cases that had their execution recorded once (*capture* part). However, the scripts generated in the *capture* part are fragile to GUI layout change, which can render entire automated test suites inept [2]. In [2] *Visual GUI Testing* uses image recognition and it is less hard-coded than Capture-and-Replay to the GUI element positioning. Such approaches compare different versions of UIs. However, the coverage of testing is never exhaustive, and a runnable version of the system under test is required. This requirement is bypassed with formal verification approaches.

In [3] a formal method to verify if a UI is a refinement of another UI is proposed, by verifying *functional* equivalence, for instance. This verification is similar to our *inclusion* verification: it verifies if a UI provides at least as much as another UI. However, they do not verify different levels of equivalence.

Degani and Heymann [7] used a formal approach to compare UI and user models (which reflects how users perceive the system). Their approach permits to identify if users elaborate an inadequate mental model of the system, leading them to confusion and errors. Similarly, in [13] an approach to reasoning over a device and a user model is proposed, using Symbolic Analysis Laboratory. In [6] a model inherited by LTS was also proposed, called HMI LTS, to represent system and user models. It mainly derives the kind of actions that can trigger state changing. However, these approaches do not reason over different versions of a UI.

Other formal approaches were proposed [15, 17, 18, 14] to reason over UIs. However, instead of comparing UIs, they verify the satisfiability of UI model with respect to some properties, using model checking and/or theorem proving.

CONCLUSION

We present an approach to comparing UIs using equivalence checking. Two UI aspects are covered: interaction capabilities and appearance. We show whether two UIs are equivalent, equivalent modulo some features, included one in the

other, or neither one. The approach was successfully applied to a case study in the nuclear power plant domain.

One limitation of the approach is that it relies on the ISLTS representation of the model. Depending on the abstractions, the number of ISLTS states/transitions may largely increase. Alternatives exist in CADP to handle big models, avoiding state space explosion (e.g. compositional verification), but they need further investigation with larger case studies.

The approach was thought to compare plastic UIs, in which some similarity between UIs exists. In the future, we plan to study when and how to apply the approach, for instance during the design process to the respective task, abstract, concrete and final UI models [4]. The approach could also be valuable to compare UI versions along product evolutions. More generally, it can also be used to compare any UIs. In such case, a large use of abstraction techniques is required. These perspectives show that equivalence is promising for UI comparison in any context.

ACKNOWLEDGMENTS

This work is funded by the French Connexion Cluster (Programme d'Investissements d'avenir / Fonds national pour la société numérique / Usages, services et contenus innovants). We warmly thank Frédéric Lang and Hubert Garavel, researchers at INRIA Rhône-Alpes, for their strong contribution to the work.

REFERENCES

1. Bauersfeld, S. GUIdiff - A Regression Testing Tool for Graphical User Interfaces. In *ICST*, IEEE (2013), 499–500.
2. Borjesson, E., and Feldt, R. Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry. In *ICST*, IEEE (2012), 350–359.
3. Bowen, J., and Reeves, S. Refinement for user interface designs. *Electronic Notes in Theoretical Computer Science 208* (2008), 5–22.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.
5. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., McKinty, C., Powazny, V., Lang, F., Serwe, W., and Smeding, G. Reference Manual of the LNT to LOTOS Translator (Version 6.1), 131 pages. 2014.
6. Combéfis, S. *A Formal Framework for the Analysis of Human-Machine Interactions*, vol. 459. Doctoral thesis, Universit catholique de Louvain, 2013.
7. Degani, A., and Heymann, M. Formal Verification of Human-Automation Interaction. *Human Factors 44* (2002), 28–43.
8. Findlater, L., and McGrenere, J. Comprehensive user evaluation of adaptive graphical user interfaces. In *Workshop on Usable AI at CHI* (2008), 41–44.
9. Gajos, K. Z., Czerwinski, M., Tan, D. S., and Weld, D. S. Exploring the Design Space for Adaptive Graphical User Interfaces. In *AVI* (2006), 201–208.
10. Garavel, H., and Lang, F. SVL : a Scripting Language for Compositional Verification. Research Report RR-4223, 2001.
11. Garavel, H., Lang, F., Mateescu, R., and Serwe, W. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *International Journal on STTT* (2013), 89–107.
12. Jung, H., Lee, S., and Baik, D.-K. An Image Comparing-based GUI Software Testing Automation System. In *SERP* (2012), 318–322.
13. Masci, P., Rukšėnas, R., Oladimeji, P., Cauchi, A., Gimblett, A., Li, Y., Curzon, P., and Thimbleby, H. The benefits of formalising design guidelines: A case study on the predictability of drug infusion pumps. *Innovations in Systems and Software Engineering* (2013), 1–21.
14. Navarre, D., Palanque, P. A., Ladry, J.-F., and Barboni, E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM TOCHI* (2009), 18:1–18:56.
15. Oliveira, R., Dupuy-Chessa, S., and Calvary, G. Formal Verification of UI Using the Power of a Recent Tool Suite. In *ACM SIGCHI symposium on EICS* (2014), 235–240.
16. Park, D. Concurrency and Automata on Infinite Sequences. In *GITCS*, Springer-Verlag (1981), 167–183.
17. Paternó, F. Formal Reasoning About Dialogue Properties With Automatic Support. *Interacting with Computers* (1997), 173–196.
18. Sousa, M., Campos, J., Alves, M., and Harrison, M. Formal Verification of Safety-Critical User Interfaces: a space system case study. In *Formal Verification and Modeling in Human Machine Systems: AAAI Spring Symposium*, AAAI Press (2014), 62–67.
19. Thevenin, D., and Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. *Proc Interact* (1999), 110–117.
20. van Glabbeek, R. J., and Weijland, W. P. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM* (1996), 555–600.
21. Vanderdonckt, J., Calvary, G., Coutaz, J., and Stanciulescu, A. *Multimodality for Plastic User Interfaces: Models, Methods, and Principles*. 2008, chapter 4, 61–84.