Research article

Raquel Oliveira*, Sophie Dupuy-Chessa, Gaëlle Calvary

Verification of Plastic Interactive Systems

DOI 10.1515/icom-2015-0036

Abstract: Interactive systems have largely evolved over the past years. Nowadays, different users can interact with systems on different devices and in different environments. The user interfaces (UIs) are expected to cope with such variety. Plastic UIs have the capacity to adapt to changes in their context of use while preserving usability. Such capability enhances UIs, however, it adds complexity on them. We propose an approach to verifying interactive systems considering this adaptation capability of the UIs. The approach applies two formal techniques: model checking, to the verification of properties over the system model, and equivalence checking, to compare different versions of a UI, thereby identifying different levels of UI equivalence. We apply the approach to a case study in the nuclear power plant domain in which several UI are analyzed, properties are verified, and the level of equivalence between them is demonstrated.

Keywords: Equivalence Checking, Formal Verification, Interactive Systems, Model Checking, Plasticity, User Interfaces

1 Introduction

The advent of ubiquitous computing and the increasing variety of platforms and devices change user expectations in terms of user interfaces. Systems should be able to adapt to their *context of use*, i. e., the *platform* (e. g. a PC or a tablet), the *users* who interact with the system (e. g. administrators or regular users), and the *environment* in which the system runs (e. g. a dark room or outdoor). The capacity of a UI to withstand variations in its context of use while preserving usability is called *plasticity* (Thevenin and Coutaz 1999).

Plasticity provides users with different versions of a UI. Although it enhances UI capabilities, plasticity adds complexity to the development of user interfaces: consistency between multiple versions of a given UI (one for each context of use) should be ensured; functionalities that are critical to the system should be preserved in all UI versions; and ergonomic properties the UI is expected to fulfill with should be satisfied in all its versions. Given the large number of possible versions of a UI, it is time-consuming and error prone to check these requirements by hand. Some automation must be provided to verify plasticity. This complexity is further increased when it comes to UIs of safety-critical systems. Safety-critical systems are systems in which a failure has severe consequences (e. g. death or injury to people, environmental harm, loss or damage to equipment). Several issues have been reported in the safety-critical domain due to bad UIs (e. g., in avionics (Degani and Heymann 2002), in radiation therapy machines (Leveson and Turner 1993), in infusion pumps to deliver drugs in hospitals (Thimbleby 2010), etc.). UIs are now expected not only to provide correct, intuitive, non-ambiguous and adaptable means for users to accomplish a goal, but also to cope with safety requirements aiming to make sure that systems are reasonably safe before they enter the market.

Several techniques to ensure quality of systems exist, such as *simulation, testing, code reviews, static analysis, formal verification*, etc (Garavel and Graf 2013) formal. Formal verification is suitable for safety-critical systems (Lutz 2000). It consists in the application of techniques that are strongly rooted in mathematics to reason over a model of the system, providing a rigorous way to perform system verification. Our contribution is a global approach to verify safety-critical interactive systems provided with plastic UIs using formal methods. Using a powerful tool-support, our approach permits to verify sets of properties over a system model, as well as to compare different versions of UIs, showing to which extent they have the same interaction capabilities and appearance.

2 Background in Verification

Formal verification requires the use of formal models. Formal models are system descriptions in a very precise

***Corresponding author: Raquel Oliveira,** University of Grenoble Alpes, Grenoble, France, e-mail: raquel.oliveira@imag.fr
Sophie Dupuy-Chessa, Gaëlle Calvary: University of Grenoble Alpes, Grenoble, France

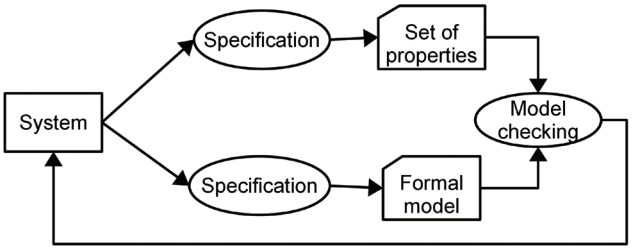

Figure 1: Model checking.

language which, unlike natural human languages, does not allow for double meanings. Such unambiguity allows formal models to be subject to deeper analyses using several formal verification techniques. This permits, for instance, the simulation of the system, the early verification of properties or the detection of inconsistencies in requirements. Examples of formal verification techniques are model checking, and equivalence checking, which we use in this work.

Model checking (Figure 1) permits to verify whether a model satisfies a set of requirements, which are specified as properties. A property is a general statement expressing an expected behavior of the system. In model checking, a formal model of the system under analysis is needed to be created, which is afterwards represented as a finite-state machine (FSM). This FSM is then subject to exhaustive analysis of its entire state space to determine whether the properties hold or not. The analysis is fully automated and the validity of a property is always decidable (Clarke et al. 1983). Expected properties should be also formalized, which in is this case is done by means of temporal logics. The analysis is mainly supported by the generation of *counter-examples* when a property is not satisfied. A counter-example can be a set of steps that when followed, by interacting with the system, leads to a state in which the property is false. The results of the analysis permits a refinement of the modeled system.

Rather than verifying the satisfiability of properties, *equivalence checking* (Figure 2) permits to formally prove whether two representations of the system exhibit exactly the same behavior or not. In order to verify whether two systems are equivalent or not, a model of each system should also be created, and then both models are compared in the light of a given equivalence relation. Several equivalence relations are available in the literature (e. g. *strong bisimulation* (Park 1981) and *branching bisimulation* (van Glabbeek and Weijland 1996)). Which relation to choose depends on the level of details of the model and the verification goals. The results of the analysis also permits a refinement of the modeled systems.

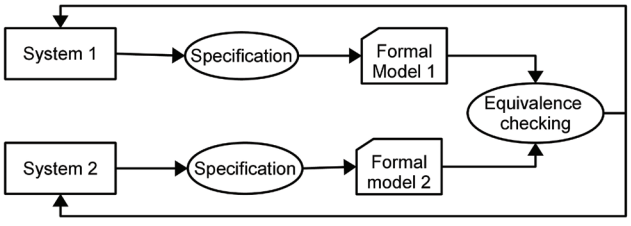

Figure 2: Equivalence checking.

Both model checking and equivalence checking are techniques that reason over the LTS (Labeled Transition System) representation of the system. A LTS is a graph composed of states and transitions between states. Transitions between states are triggered by actions. Intuitively, a LTS represents all possible evolutions of a system modeled by a formal model.

These two verification techniques constitute the elements used in our approach to verify plastic interactive systems.

3 Global Approach

This section introduces our global approach to verifying interactive systems, as well as the rationale for the design and technical choices.

3.1 Overview

We propose a global approach (Figure 3) to assess quality of safety-critical interactive systems with plastic UIs. We integrate both model checking and equivalence checking formal techniques. We use model checking to verify properties over the formal specification of the interactive system, and equivalence checking to compare formal specifications of the system in different contexts of use.

The formal verifications are performed over the ISLTS (Interactive System LTS) representation of the specifications. We derive ISLTS from standard LTS in order to represent both UI interaction capabilities and appearance in one single model (Oliveira et al. 2015a).

Model checking and equivalence checking can be used independently or in an integrated way. Independently, disregarding the UI adaptation capabilities of the interactive system, we propose the usage of model checking to verify properties over the system formal model. Considering that the system UIs are provided with plasticity capabilities, we propose a technique to compare different versions of the UIs by means of equivalence checking.

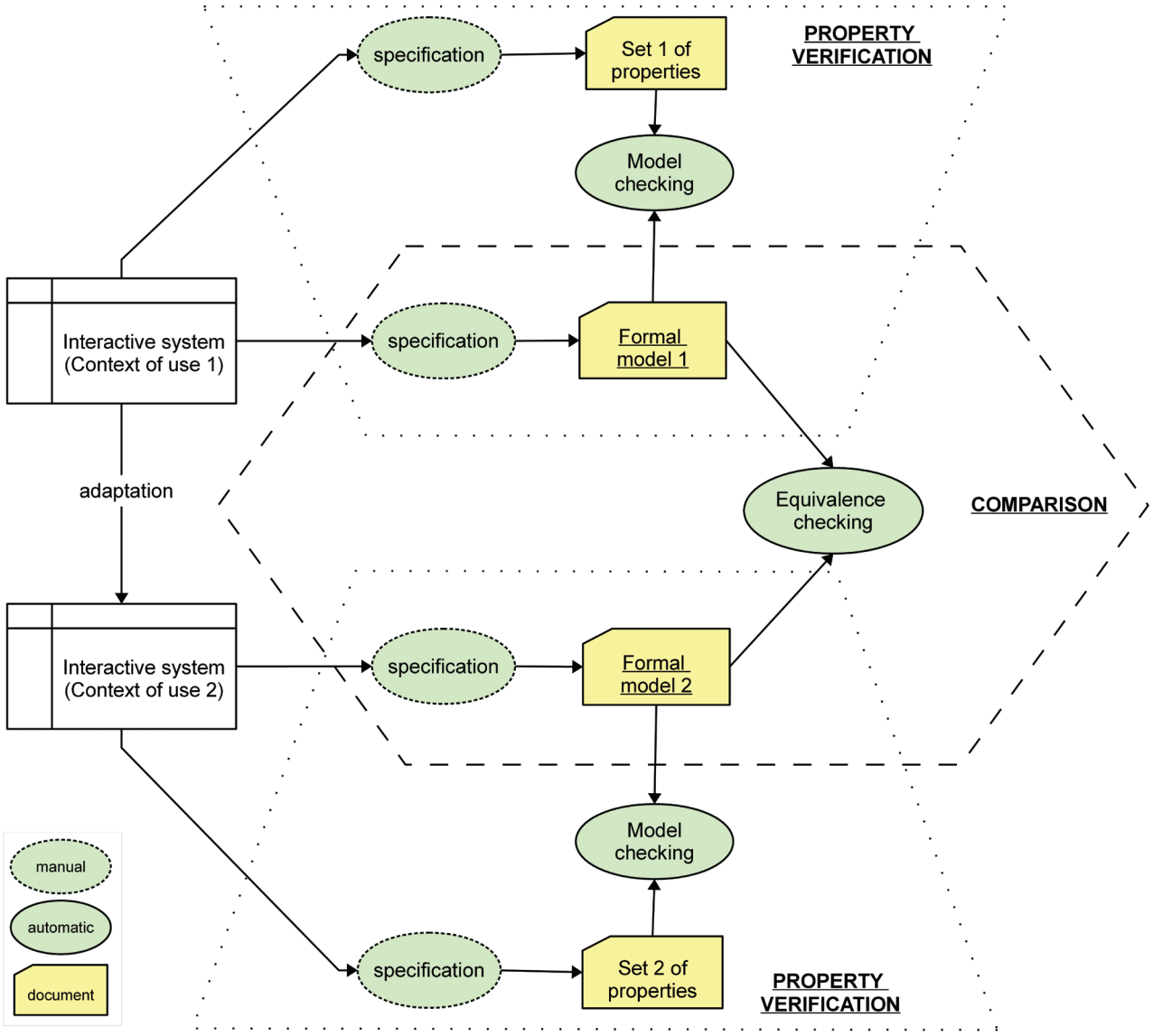

Figure 3: Global approach to verifying interactive systems with plastic UIs.

Optionally, model comparison can be integrated with property verification. Before checking equivalence of the formal models, checking a set of properties over the models can guarantee that they cope with a certain level of quality, increasing the relevance of the equivalence checking results. In case the models under comparison are expected to satisfy the same set of properties, the verification can be reduced to check one model of the interactive system, and to verify the equivalence between both models: if one model satisfies the set of properties, and this model is equivalent to the other one, then the second model also satisfies the properties. If each model is expected to satisfy different properties, due to particularities of their context of use, the approach is fully performed: model checking each formal model with respect to their set of properties, followed by a equivalence verification of the models.

3.2 Design Choices

In order to obtain more reusable results, system models are represented following the ARCH architecture (Bass et al. 1991). Architectural models provide a means to structure systems, using the principle of separation of concerns. In ARCH, systems are decomposed in five main components: the functional core, the functional core adaptor, the logical presentation, the physical presentation and the dialog controller (i. e., the blue boxes in Figure 4). We use the simplified version of ARCH (i. e., the dashed boxes in Figure 4),

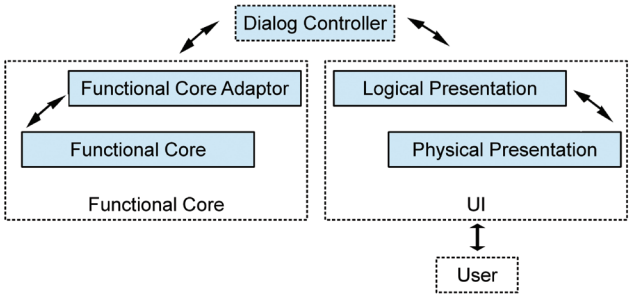

Figure 4: ARCH architecture usage in formal modeling.

in which the major functional components of an interactive system are represented: a functional core component, which groups the two components of ARCH, the UI component, which group two ARCH components, and the dialog controller component. We propose to take into account this separation of concerns when creating the formal models of the system. In addition, in our approach a user module is created. Figure 4 illustrates how the formal model is organized: each dashed box represents one or more parts of the formal model. The formal model reflects the fact that users interact only with the UIs, not having access neither to the functional core of the system nor to the dialog controller.

3.3 Technical Choices

The application of the approach is supported by several formal languages and tools from the CADP[1] toolbox (Construction and Analysis of Distributed Processes) (Garavel et al. 2013). The choice of the toolbox was mainly motivated by its maturity, continuous evolution and support, and the numerous included tools. **CADP** is a toolbox for verifying asynchronous concurrent systems: systems whose components may operate at variable speeds, without a global clock to synchronize them. Such components communicate and exchange information from time to time by channels. Asynchronous systems suit well the modeling of human-machine interactions: it permits the components that describe the users, the functional core and the UIs to evolve in time at different speeds, which reflects well the unordered sequence of interactions that can take place in human-machine interactions.

We use **LNT** (Champelovier et al. 2014) formal language to specify the system formal models, which is one of the input specification languages supported by CADP. LNT is a specification language derived from ELOTOS ISO standard. It improves LOTOS, and can be translated to LOTOS automatically. LNT has several advantages over LOTOS, notably the user friendliness and the richer data type definition.

CADP can generate a ISLTS from LNT formal models, which can be subject to operations such as minimization (also called reduction), abstraction, comparison, deadlock/livelock detection, etc., by means of a scripting language called **SVL**[2] (Script Verification Language) (Garavel and Lang 2001).

In order to formalize the expected properties of the interactive systems, we use **MCL**[3] (Model Checking Language) (Mateescu and Thivolle 2008). MCL is an enhancement of the modal μ-calculus, a fixed point-based logic that subsumes all other temporal logics, aiming at improving the expressiveness and conciseness of formulas (Mateescu and Thivolle 2008). Specifically, MCL adds data-handling mechanisms; it adds a fairness operator; it contains quantifiers over finite data domains and constructors inspired from functional programming (e. g. let, if-else, case, while, repeat, etc.) (Mateescu and Thivolle 2008).

This global approach of verification was applied to a case study in the nuclear-plant domain, in which a set of properties was verified and several versions of a user interface were compared to each other.

4 Case Study

Our case study concerns a control room system of a nuclear power plant that provides an overview of the plant state (Chériaux et al. 2012). It notifies the operator about all unexpected events in the plant. The main UI contains four zones (Figure 5, in French):

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 reactor parameters.
3. At the bottom, the *Parameter* ("Paramètres") zone displays various reactor parameters (e. g. the pressure), each one represented by a widget containing: the parameter name, its current value, a curve with

1 http://cadp.inria.fr

2 http://cadp.inria.fr/man/svl.html

3 http://cadp.inria.fr/man/evaluator4.html, section "Overview of the MCL Language"

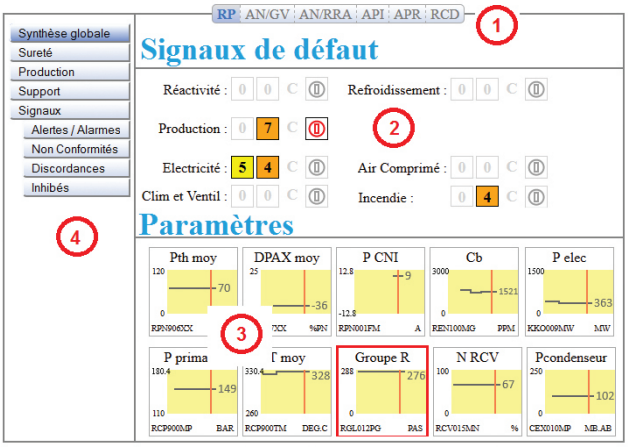

Figure 5: A nuclear power plant control room system. Main UI of the system – PC Version.

the value evolution over time, a minimum/maximum value boundaries, the sensor that 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.

4. On the left, users 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.

5 Verification of Properties

In order to verify a set of ergonomic properties over the system, both the system and the properties should be formalized. Nowadays the system model is manually written, in contrast to the work proposed in (Paternó 1997), that generates a LOTOS specification directly from a task model. In our case, a task model per se does not contain sufficient information to permit automatic generation of the formal model. It turns out that our formal model covers the user interface behavior and some aspects of the functional core. For this reason, the formal model is written manually, to be as realist as possible.

Once the formal specification is created, one can verify a set of properties on it. Our approach suggests the usage of ergonomic guidelines to support the identification of the desired properties. A lot of work has been done to guide the identification of user interface properties. In our approach, the usability properties from the framework proposed in (Abowd et al. 1992) were chosen. All the identified properties are classified as *robustness* properties, which refer to the possibility of navigating through the observable states of the system.

We write the properties in MCL language. For example, the property:

"From any UI, one can always go directly to the main UI (i. e. without passing through any other UI)"

is expressed in MCL in the following way:

$[true^*] \langle (not(UI))^*. \text{'GLOBAL\_SYNTHESIS'} \rangle true$

and may be read as:

[From every reachable state] ⟨ there exists a sequence of steps, not passing through any UI, and leading to the GLOBAL_SYNTHESIS UI ⟩

This property ensures that, in all user interfaces, there is always the possibility to come back to the main UI with one single user interaction, i.e. without the need to access intermediate UI before. Once the desired properties are formalized, they are verified using the model checker of CADP toolbox.

5.1 Discussion

The key enhancement brought by our approach is the usage of a more powerful support (Oliveira et al. 2014). The user-friendliness of the LNT language decreases the learning curve of designers in formal methods, and it decreases the required labor time of writing a formal specification of the system, enabling one to more quickly take advantages of formal methods. The rich data type definitions of LNT permits more realistic UI models, thus widening the capabilities of verification, covering verifications on the data type of the UI fields, for instance.

The use of MCL to formalize the properties is another advantage of our approach. MCL permits to identify, for example, the existence of complex unfair (infinite) cycles in the ISLTS generated from the formal model. An unfair cycle is an infinite sequence made by the concatenating sub-sequences satisfying the formula (Mateescu and Thivolle 2008), e. g. a sequence of actions over the user interface that once started loops forever.

The set of tools is also important to support formal analysis. Rather than developing our own tool to perform formal verification, we work in collaboration with the authors of CADP toolbox. In particular, CADP has continuously evolved over the past years. By taking advantage of its new capabilities, it is now possible for example to perform *compositional verification* on individual processes of the model, enabling to handle much larger state spaces. In practice, bigger models can be handled, so that we can consider more complex UIs and more realistic UI models.

5.2 Validation

We use LNT to write the formal specification of the case study. Hand-written modeling adds subjectivity to the formal model. To avoid this, the formal models were validated with an expert in the nuclear-plant domain. Following ARCH, the formal specification contains 6 LNT modules: the *plant_status* and *menu* modules describe some zones of the UI; the *reactor* and *generate_signals* describe some functions of the functional core; the *selection* module describes the dialog controller; and, beyond ARCH, an additional module called *user* was added, to describe part of the user behavior. The whole specification contains 15 modules in total, and 2462 lines of LNT code.

CADP tools were used to generate the ISLTS from the LNT specification and to verify the properties over the model using model checking. In particular, the EVALUATOR 4.0 model checker was used, as well as the OCIS[4] tool (Open / Caesar Interactive Simulator) for step-by-step simulation with backtracking, permitting to explore all the possible executions of the model.

Nine properties were identified and written in MCL (Oliveira et al. 2014). All properties were satisfied over the formal model, which is an evidence that the modeled system satisfies such properties. This modeling of the case study was the basis for the verification of plastic user interfaces.

6 Comparison of User Interfaces

The control room system has the need to adapt to different contexts of use. For instance, to make operators mobile, a tablet version of the UIs could be provided. Our approach permits to identify different levels of equivalence between different versions of a UI.

6.1 UI Versions

In (Vanderdonckt et al. 2008) the dimensions of UI adaptation were studied and the problem space of plastic UIs was defined, in which seven dimensions were identified. The *Adaptation Means* dimension, which refers to the means used for UI adaptation, is our primary concern in this paper. Two different means were identified: UI *re-molding* denotes any UI reconfiguration that

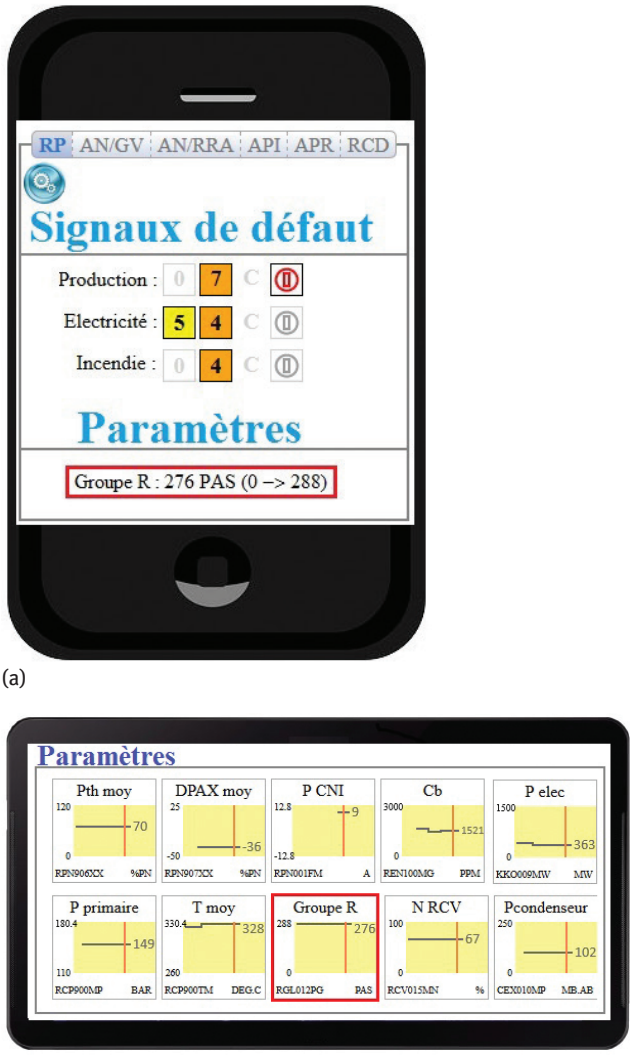

Figure 6: UI *platform* adaptation, (a) Smartphone UI, (b) Tablet UI.

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.

Figure 6a illustrates an example of re-molding: the control room UI is adapted to the target platform (a Smartphone). While on the PC (Figure 5) 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. Finally, 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.

Figure 7a and 7b illustrate another example of re-molding, in which the UI is adapted to the target user.

4 http://cadp.inria.fr/man/ocis.html

(a) (b)

Figure 7: UI *user* adaptation, (a) Training mode, (b) Expert mode.

In *training mode* (Figure 7a), the following elements are added: 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, Production, or Support. In *expert mode*, all this guidance is removed.

A Tablet version (Figure 6b) 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. UI *interaction capabilities* concern the ways users can interact with the UI (and, reversely, how the UI reacts to this interaction). UI *appearance* concerns the elements present in the UI (where they are presented, in which color, etc.).

Different versions of a UI can diverge at several levels. Such levels should be taken into account when comparing the UI versions. Our approach permits to identify the level of equivalence between UIs, and it covers four levels of equivalence: equivalent UIs, non-equivalent UIs, equivalent modulo "X" UIs and included UIs. A general framework formalizing these four levels of UI equivalence is provided in (Oliveira et al. 2015a). The analysis is supported by three abstraction techniques, which are explained in the following.

6.2 Equivalent User Interfaces

In the light of our UI comparison formal framework (Oliveira et al. 2015a), two UIs are **equivalent** when, at a certain level of abstraction, they present the same interaction capabilities and appearance, i. e., whenever the user can perform the same actions in both UIs, both UI versions respond with the same feedback, and the same information is displayed on both UIs.

The equivalence which can be shown between two UIs varies from strongly equivalent to weaker equivalent. This is defined by the level of abstraction of the UI models. 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 in the ISLTS (i. e. τ), and can be ignored, although they are still present in the UI model. We call this abstraction technique an **omission**. This abstraction is useful, for instance, when users are provided with a functionality activated in different ways in the UIs: for example, two UIs that have menus with the same options, but in one UI the menu is always unfolded and in the other UI it is folded. Omission permits the action of unfolding the menu to be ignored when comparing the UIs. When UI actions are bypassed in the analysis, a weaker equivalence between the UIs is shown.

Another abstraction technique we propose is called **generalization**, and it concerns only UI appearance (not interaction capabilities). This technique permits the representation of the UI appearance in different levels of details, allowing information to be summarized into a more general representation.

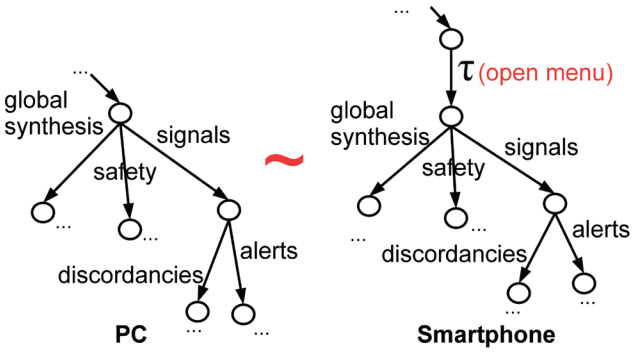

Figure 8: ISLTS fragments of PC and Smartphone UIs.

To illustrate the approach, consider a UI adaptation according to the platform, to which re-molding was applied: PC (Figure 5) and Smartphone (Figure 6a). Regarding the UI interaction capabilities, the menu in both UIs 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 (Figure 8). 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: a user action that does not have an impact on the 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.

Concerning the UI appearance, both signals and parameters are displayed in the same zones. For this analysis, 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. Figure 9 illustrates such differences in an ISLTS fragment. Both frames on top represent 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").

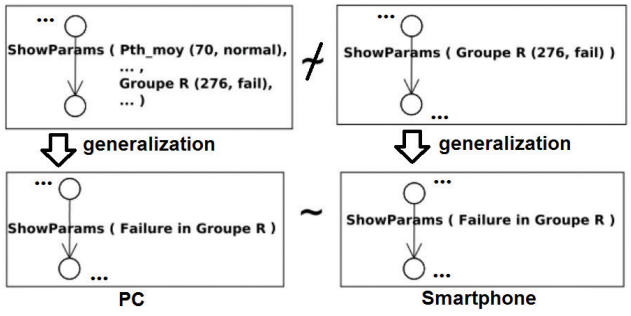

Figure 9: *Generalization* in an ISLTS.

Generalization abstraction permits to generalize actions containing detailed information into less detailed actions (i. e. the "Failure in *x*" renamed action at the bottom frames). Using *generalization* and *omission* abstractions, the PC UI model and the Smartphone UI model are equivalent.

6.3 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. Two UIs are **equivalent modulo "X"** when, discarding the functionality "X" from the analysis, both UIs are equivalent.

Elimination abstraction can be used to bypass a UI functionality, permitting the removal of elements in the UI model before performing the analysis. UI interaction capabilities can be discarded, together with the UI appearance once the action is executed. Contrary to *omission*, in which the elements are still present in the model and are ignored, here the elements are removed.

To illustrate the technique, consider a UI adaptation according to the user expertise, to which re-molding was applied: Training Mode (Figure 7a) and Expert Mode (Figure 7b). Regarding the appearance, there are several differences between the two UIs. We use *generalization* to represent differences 2, 3 and 4 (Figure 7a).

Concerning the UIs interaction capabilities, the Training-mode UI contains one additional navigation aid: a breadcrumb trail (i. e. the difference 1 in Figure 7a). We set the equivalence checking to be done disregarding this feature. We use *elimination* abstraction (Figure 10) to search (in the ISLTS) actions corresponding to the breadcrumb trail (i. e. the *bct_* pattern). Once a match occurred,

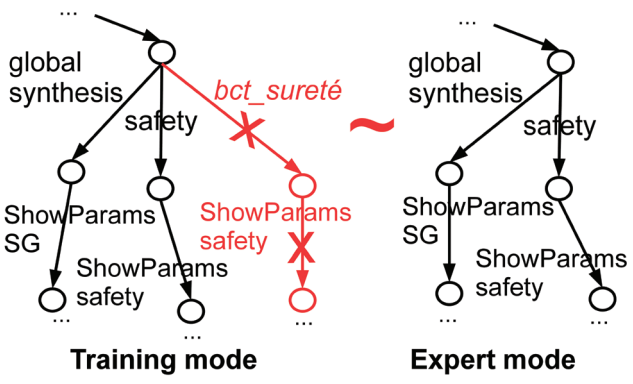

Figure 10: *Elimination* in an ISLTS.

all the successor states (and transitions) were eliminated in cascade from the ISLTS.

Using *generalization* (for the items n. 2, 3, and 4 of Figure 7a) and *elimination* (for the item n. 1), the Training and Expert UI models are equivalent modulo the bread-crumb trail.

6.4 Non-Equivalent User Interfaces

There are cases in which two UIs present a large number of divergences, and eliminate all the divergences compromises the usefulness of the results. In this case, the UIs are **not equivalent**.

6.5 Inclusion of User Interfaces

Two UIs can also relate with each other by the **inclusion** relation. Intuitively, a given user interface U_1 includes another user interface U_2 whenever the former contains *at least* all interaction capabilities (and the appearance) of the latter.

To illustrate, consider a UI adaptation according to the target platform, to which redistribution was applied: PC (Figure 5) and Tablet versions (Figure 6b). Regarding UI interaction capabilities, the functionalities that permit user interactions are available only on the PC (i. e. the menu and the plant status selection). With respect to appearance, the UIs also diverge: the Tablet version does not contain the plant status, the reactor signals and the menu zones.

The divergences of these two UIs are too large to consider the use of *elimination* abstraction. In this case, we apply no abstraction techniques, and the two UI models are 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 show that the PC version contains at least all functionalities (regarding interaction capabilities and appearance) of the Tablet version. The PC-version UI model includes the Tablet-version UI model (i. e. *Tablet_model* ≤ *PC_model*).

6.6 Discussion

The abstraction techniques introduced here support UI model comparison. The principle is to first create abstract models of the UIs, used afterwards to perform equivalence checking. Figure 11 illustrates the different levels of equivalence between two UI models. 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, numerous divergences are present within the UI versions. The challenge is to verify equivalence between the UI models in spite of these divergences. Abstraction techniques provide means to do that, and weaker equivalence relations between the models can be shown. The more abstractions are applied to the models, the weaker the equivalence between the models becomes. Transversally, the inclusion between two UI can be verified at any level of abstraction. The results of the comparison allow a refinement of the formal models and/or the real UIs.

6.7 Validation

A LNT formal model was manually written for the 5 contexts of use presented in this paper (i. e., PC, Smartphone, Tablet, Training and Expert Mode). The abstract criteria were implemented using SVL language. Table 1 summarizes the number of lines of LNT code and the ISLTS size. The case study shows that the approach scales well (Oliveira et al. 2015a). It was initially designed for one context of use (PC), later extended to 5 contexts of use. Each formal model contains 3 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 (< 3 h).

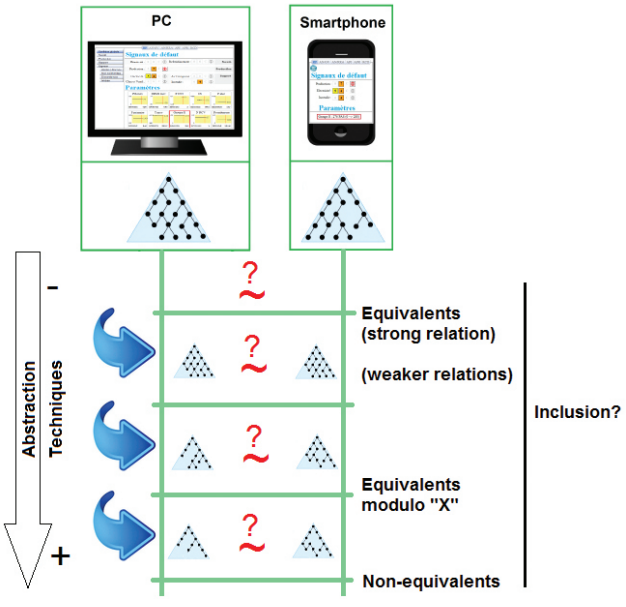

Figure 11: Different levels of equivalence between UI models.

Table 2 illustrates the summary of the comparisons, where *O* indicates the number of omissions done, *G* the number of generalizations, *E* the number of eliminations, and the last column indicates the comparison time. The PC and the Smartphone versions were shown equivalent, the Training and the Expert versions were shown equivalent modulo the breadcrumb trail feature and the Tablet version was shown included in the PC version. The comparison of the ISLTS was done using BCG_CMP[5] and BISIMULATOR[6] tools of CADP.

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

Table 2: Summary of the comparisons.

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

7 Related work

Property verification by model checking has been proposed several times in the past years (Navarre et al. 2009, Sousa et al. 2014, Paternó 1997, Oliveira et al. 2014). But none of them cover plastic UIs. Similar to our UI comparison approach, existing approaches compare UIs in different ways. Some are supported by classical testing (Bauersfeld 2013, Jung et al. 2012) and others by formal methods (Bowen and Reeves 2008). In the following, we compare such approaches according to the following criteria: (1) *model coverage*: to be as representative as possible, the model should cover aspects of the users, the UIs and the functional core; (2) *application to the nuclear plant domain*: we will analyze whether the approach was applied to this domain or not; (3) *scalability*: whether it scales well for real-life applications.

In (Bauersfeld 2013) *regression testing* is applied over the new version of a UI, providing a list of the detected differences. The approach was applied to several case studies and it is tool-supported (Bauersfeld et al. 2014), indicating that it scales well for real-life systems. However, no application to safety-critical systems was found, and only the UIs are covered in the verification. In (Jung et al. 2012) *Capture-and-Replay* technique was used to perform regression testing of UIs. However, the scripts generated in the *capture* part are fragile to GUI layout change, which can render entire automated test suites inept. The paper provides no evidence that the approach scales well. No application of the approach to safety-critical systems was found either, and only the UIs are covered. In (Bowen and Reeves 2008) a formal technique to verify if a UI is a refinement of another UI is proposed, which covers the modeling of users, UIs and the functional core. The approach verifies *functional* equivalence, which is similar to our *inclusion* verification: it verifies if a UI provides at least as much as another UI. However, it does not verify different levels of equivalence, and it was not applied to safety-critical systems. Besides, the verification is performed with no automation, by manual inspection of the UI models, giving no evidence that the approach scales well for larger applications.

Table 3 summarizes these model-based approaches to verify interactive systems by comparing different versions of the system, the means to represent the system, the technique used for comparison, the chosen tool and the criteria we used to analyze them. Specifically, no approach so far covered plastic UIs. Our approach aims at verifying safety-critical systems, and it was applied to a realistic case study in the nuclear-plant domain, which shows that it scales well for real-life applications. Besides, it covers the modeling and verification of the user, the UIs and the functional core.

8 Conclusion

We presented a global approach to verifying safety-critical interactive systems provided with plastic UIs, in which sets of properties can be verified over a formal specification of the system by means of model checking, and different versions of plastic UIs can be compared with each other by means of equivalence checking. In the UI comparison, two UI aspects are covered: interaction capabilities

5 http://cadp.inria.fr/man/bcg_cmp.html
6 http://cadp.inria.fr/man/bisimulator.html

Table 3: Summary of approaches that compare different versions of the system model.

Authors	Verification			Criteria				
	Language	Technique	Tool	Model coverage			Applied	Scalability
				user	UI	core		
Bauersfeld *et al.*	trees	modelbased testing	GUITest, GUIDiff, Rogue / TESTAR		√		noncritical	yes
Jung *et al.*	images	modelbased testing	a prototype		√		noncritical	no evidence
Bowen and Reeves	Z, μCharts, FSM	manual model inspection	–	√	√	√	noncritical	no evidence
Oliveira *et al.*	LNT	model checking, equiv. checking	CADP	√	√	√	nuclearplants	yes

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.

We have started investigations on other techniques to verify plastic user interfaces (Oliveira et al. 2015b). For the moment, we have mainly explored the comparison of UIs. Other possibilities are the verification of properties common to all the UI versions and the verification of properties over the adaptation engine implementing the transformation rules of the UIs. In the future, we plan to deeper investigate these two alternatives, in particular in terms of the description of the adaptation engine logic.

We also 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. 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.

Acknowledgment: 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

Abowd G. D., J. Coutaz and L. Nigay. 1992. Structuring the space of interactive system properties. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, pages 113–129, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co. ISBN 0-444-89904-9. URL http://dl.acm.org/citation.cfm?id=647103.717569.

Bass L., R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard and M. R. Szezur. 1991. The ARCH Model: Seeheim Revisited. In *User Interface Developpers' Workshop*.

Bauersfeld S. 2013. GUIdiff – A Regression Testing Tool for Graphical User Interfaces. In *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, ICST '13, pages 499–500, Washington, DC, USA. IEEE Computer Society. ISBN 978-0-7695-4968-2. 10.1109/ICST.2013.84. URL http://dx.doi.org/10.1109/ICST.2013.84.

Bauersfeld S., T. E. J. Vos, N. Condori-Fernandez, A. Bagnato and E. Brosse. 2014. Evaluating the TESTAR Tool in an Industrial Case Study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 4:1–4:9, New York, NY, USA. ACM. ISBN 978-1-4503-2774-9. 10.1145/2652524.2652588. URL http://doi.acm.org/10.1145/2652524.2652588.

Bowen J. and S. Reeves. Apr. 2008. Refinement for User Interface Designs. *Electron. Notes Theor. Comput. Sci.*, 208:5–22. ISSN 1571-0661. 10.1016/j.entcs.2008.03.104. URL http://dx.doi.org/10.1016/j.entcs.2008.03.104.

Champelovier D., X. Clerc, H. Garavel, Y. Guerte, C. McKinty, V. Powazny, F. Lang, W. Serwe and G. Smeding. aug 2014. Reference Manual of the LNT to LOTOS Translator (Version 6.1). INRIA/VASY and INRIA/CONVECS, 131 pages.

Chériaux F., D. Galara and M. Viel. 2012. *Interfaces for nuclear power plant overview. In 8th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human-Machine Interface Technologies (NPIC & HMIT 2012): Enabling the Future of Nuclear Energy, NPIC & HMIT 2012, pages 1002–1012. Curran Associates, Inc., 2012.* ISBN 978-1-627-48015-4.

Clarke E. M., E. A. Emerson and A. P. Sistla. 1983. Automatic Verification of Finite State Concurrent System Using Temporal

Logic Specifications: A Practical Approach. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '83, pages 117–126, New York, NY, USA. ACM. ISBN 0-89791-090-7. 10.1145/567067.567080. URL http://doi.acm.org/10.1145/567067. 567080.

Degani A. and M. Heymann. 2002. Formal Verification of Human-Automation Interaction. *Human Factors*, 44:28–43.

Garavel H. and S. Graf. 2013. Formal methods for safe and secure computer systems. *Federal Office for Information Security*.

Garavel H. and F. Lang. 2001. SVL: A Scripting Language for Compositional Verification. In *Proceedings of the IFIP TC6/ WG6.1–21st International Conference on Formal Techniques for Networked and Distributed Systems*, FORTE '01, pages 377–394, Deventer, The Netherlands, The Netherlands. Kluwer, B.V. ISBN 0-7923-7470-3. URL http://dl.acm.org/citation. cfm?id=646219.682166.

Garavel H., F. Lang, R. Mateescu and W. Serwe. 2013. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *International Journal on STTT*, 15:89–107, 2013. 10.1007/s10009-012-0244-z. URL http://hal.inria.fr/ hal-00715056.

Jung H., S. Lee and D.-K. Baik. 2012. An Image Comparing-Based GUI Software Testing Automation System. In *SERP*, pages 318–322.

Leveson N. G. and C. S. Turner. July 1993. An Investigation of the Therac-25 Accidents. *Computer*, 26(7):18–41. ISSN 0018-9162. 10.1109/MC.1993.274940. URLhttp: //dx.doi.org/10.1109/ MC.1993.274940.

Lutz R. R. 2000. Software engineering for safety: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 213–226. ACM.

Mateescu R. and D. Thivolle. 2008. A Model Checking Language for Concurrent Value-Passing Systems. In *Proceedings of the 15th International Symposium on Formal Methods*, FM '08, pages 148–164, Berlin, Heidelberg. Springer-Verlag. ISBN 978-3-540-68235-6 10.1007/978-3-540-68237-0_12. URL http://dx.doi.org/10.1007/978-3-540-68237- 0 12.

Navarre D., P. Palanque, J.-F. Ladry and E. Barboni. Nov. 2009. ICOs: A Model-Based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. *ACM Trans. Comput.-Hum. interact.,* 16(4): 18:1–18:56. ISSN 1073-0516. 10.1145/1614390.1614393. URL http://doi.acm.org/10.1145/1614390.1614393.

Oliveira R., S. Dupuy-Chessa and G. Calvary. 2014. Formal Verification of UI Using the Power of a Recent Tool Suite. In *ACM SIGCHI symposium on EICS*, pages 235–240. ISBN 978-1-4503-2725-1. 10.1145/2607023.2610280. URL http://doi.acm.org/10. 1145/2607023.2610280.

Oliveira R., S. Dupuy-Chessa and G. Calvary. 2015a. Equivalence Checking for Comparing User Interfaces. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering interactive Computing Systems*, EICS '15, pages 266–275, New York, NY, USA., ACM. ISBN 978-1-4503-3646-8. 10.1145/2774225.2774844. URL http://doi.acm. org/10.1145/2774225.2774844.

Oliveira R., S. Dupuy-Chessa and G. Calvary. 2015b. Plasticity of User Interfaces: Formal Verification of Consistency. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, pages 260–265, New York, NY, USA. ACM. ISBN 978-1-4503-3646-8. 10.1145/2774225.2775078. URL http://doi.acm. org/10.1145/2774225.2775078.

Park D. 1981. Concurrency and Automata on Infinite Sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, UK. Springer-Verlag. ISBN 3-540-10576-X. URL http://dl.acm.org/citation.cfm?id= 647210.720030.

Paternó F. 1997. Formal Reasoning about Dialogue Properties with Automatic Support. *interacting with Computers*, 9(2):173–196. ISSN 0953-5438. http://dx.doi.org/10.1016/ S0953-5438(97)00015-5. URL http://www.sciencedirect. com/ science/article/pii/S0953543897000155.

Sousa M., J. Campos, M. Alves and M. Harrison. 2014. Formal Verification of Safety-Critical User Interfaces: a Space System Case Study. In *Formal Verification and Modeling in Human Machine Systems: AAAI Spring Symposium*, pages 62–67. AAAI Press.

Thevenin D. and J. Coutaz. 1999. Plasticity of User Interfaces: Framework and Research Agenda. *Proc interact99 Edinburgh A Sasse C Johnson Eds IFIP IOS Press Publ*, 99:110–117. URL http://books.google.com/books?hl=en&lr=&id=yXehjiOd_ kkC&oi= fnd&pg=PA110&dq=Plasticity+of+User+Interf aces:+Framework+and+Research+ Agenda&ots=NTnX-sS3yHr&sig=yUEc2R_iTCZeIz119UjLjCw3_3o.

Thimbleby H. 2010. Think! Interactive Systems Need Safety Locks. *CIT. Journal of Computing and information Technology*, 18(4):349–360.

van Glabbeek R. J. and W. P. Weijland. 1996. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, pages 555–600. ISSN 0004-5411. 10.1145/233551.233556. URL http://doi.acm.org/10.1145/233551.233556.

Vanderdonckt J., G. Calvary, J. Coutaz and A. Stanciulescu. 2008. Multimodality for Plastic User Interfaces: Models, Methods, and Principles. In *Multimodal User interfaces*, pages 61–84. Springer.

Bionotes

Raquel Oliveira
University of Grenoble Alpes,
Grenoble, France
raquel.oliveira@imag.fr

RAQUEL OLIVEIRA is a PhD Candidate at two research groups in LIG ("Laboratoire d'Informatique de Grenoble"): EHCI (Engineering Human-Computer Interaction) and Convecs (Construction of Verified Concurrent Systems). LIG is an informatics research laboratory at the University of Grenoble Alpes, France. She currently works in the crossroad of three domains: human-computer interaction, formal methods and safety-critical systems. Her research interests include the usage of formal notations to the specification of interactive systems; the formalization of plastic user interfaces and transformation engines; the application of formal techniques to ensure quality of plastic user interfaces in the safety-critical systems.

Prof. Sophie Dupuy-Chessa
University of Grenoble Alpes,
Grenoble, France
sophie.dupuy@imag.fr

Sophie Dupuy-Chessa is professor at University of Grenoble Alpes. She got her PhD thesis in 2000 in the domain of software engineering, more precisely in software modeling. Then she has two post-doctoral positions: she was lecturer at University of Geneva and research scientist at Xerox Research Center Europe. She holds a authorization to steer researches (HDR) in Computer Science from the University of Grenoble in 2011. Currently, her research interest concerns model-driven engineering for human-computer interaction and information system design.

Prof. Gaëlle Calvary
University of Grenoble Alpes,
Grenoble, France
gaelle.calvary@imag.fr

Gaëlle Calvary is professor in Computer Science at Grenoble Institute of Technology since 2009, and a member of the Engineering Human Computer Interaction group since 1999. Before joining University Joseph Fourier in 1999 as assistant professor, she has served as a user interface software designer for 8 years at Thales. Her main research area is about Plasticity of User Interfaces (UI) for making UIs capable of adaptation to the context of use while preserving user-centered properties. She explores model-driven engineering as well as artificial intelligence. She also investigates worth centered design and persuasive technology. She is a member and the general secretary of the IFIP Working Group 2.7/13.4 under which auspices she co-chaired the first ACM Symposium on the Engineering of Interactive Computing Systems (EICS) held in Kingston in 2009. In France, she is the founder and cochair of a Working Group on Persuasive Technology.