2015 International Symposium on Theoretical Aspects of Software Engineering

Formal verification of plastic user interfaces exploiting domain ontologies

Abdelkrim Chebieb
ESI-LCSI
BP 68M, 16270, Oued-Smar,
Algiers, Algeria
k_chebieb@esi.dz

Yamine Ait-Ameur
INPT-ENSEEIHT-IRIT
2, rue Charles Camichel, B.P. 7122,
31071, Toulouse Cedex 7, France
yamine@enseeiht.fr

***Abstract*—This paper presents a formal model to check the interaction plasticity on a user interface (UI). An interaction is seen as an implementation (achievement) of a user task by means of interaction devices and modes of a given platform. The interaction plasticity is the ability of UI to support several interactions to perform the same task. In this work, two task models, containing different sets of interactions, are observed to check if they describe interactions that perform the same task. Each task model is represented by a labelled state-transitions system (lts). Due to the use of different interaction modes and devices, the obtained lts have different set of labels. Weak bi-simulation relationship is revisited to handle these transition systems by defining a relation on labels. This relation is borrowed from an ontology of interaction modes and devices. Model checking techniques are set up to automatically establish such a bi-simulation. A case study is used to illustrate how the approach works.**

I. Introduction

When a user interface (UI) is designed to run on several target platforms[1] and to support different interaction modes and/or devices, it can be considered as satisfying the plasticity property (qualified to be a plastic UI). Indeed, a plastic UI shall be able to switch, statically (at design time) or dynamically (at run time), from a given platform to another. The target platform may support (exactly, less or more) interaction capabilities than the original one. Plasticity is an important concern particularly for critical systems (the interaction continuity must be guaranteed when changes occur on the platform).

Two relevant characteristics relate to plasticity: adaptivity and adaptability. Adaptivity concerns a UI capable to adapt itself to a target platform, while adaptability characterizes a UI that allows a user to adapt it to a target platform. Adaptation of the UI impacts not only its software part, but also the involved devices and interaction modes, available for this platform.

A user interface handles a set of possible user tasks, realized by a given user, when interacting with the system, through this user interface by means of the available interaction devices. A task model is commonly defined to describe the interaction realizing these tasks. It highlights both the tasks to be achieved (abstract part of the task model) and how these tasks are effectively realized (concrete part of the model).

The design of a multi-platform interfaces entails building a task model for each platform. The abstract part of this task model remains identical for each platform while the concrete level is adapted to each specific platform according to its interaction modes and/or devices. *The main drawback of this approach is the need to perform task model verification and validation for each platform.* In other words, check if the different task models still describe the same task. Adaptation of task model, due to the variations on the hardware side of the UI (lose and/or gain of interaction devices in the platform), is identified as a "*main axis*" of the "*design space for UI adaptation*" in [1].

This paper presents a novel approach supporting the formal verification of the plasticity property of a user interface. This approach sets up a formal technique to check if two task models describe the same system interaction. More precisely, task models are formalized as labelled state-transitions systems (lts). Then, the equivalence of two task models is checked by establishing a bi-simulation relationship between their corresponding lts. The novelty of the approach lies in the capability to handle the comparison of lts with different labels (labels from interactions, issued from different equivalent devices, for example) thanks to the definition of a relational bi-simulation relationship. This relation exploits the relations between labels, borrowed from an ontology of interaction devices and modes. These relations make it possible to rewrite the lts to get new lts, defined on a common set of labels on which it becomes possible to check classical bi-simulation.

This paper is structured as follows. The next section presents a synthetic state of the art of the HCI plasticity, related to our work. We give a vision of the plasticity as a knowledge domain to justify the use of *ontologies*. Then our formal model to handle plasticity, based on the comparison of task models, modeled as labelled transition systems and the bi-simulation relationship is presented. Finally a case study, illustrating how the proposed approach works is sketched.

II. State of the art

A. Design of Plastic UI

Studying plasticity of user interfaces has drawn the attention of researchers. [2] records that several plastic UI design approaches focused on the software parts of a user interface, at the expense of interaction and dialogue parts. Below, we briefly review significant works achieved to handle plasticity.

The earliest approaches to address user interface adaptation, like ACE [3], FRUITS [4] and Multi-modal Widgets [5],

[1]The word platform is used to characterize the system on which the described UI is available. It gathers the software part, the hardware devices and the offered interaction capabilities of this system.

proposed to implement adaptation capabilities directly at software level. The leading idea consists in embedding the various characteristics of interaction modalities and devices present on different platforms inside a set of interface components, used to implement the UI.

Adaptation at runtime was addressed by techniques, handled by ToolGlasses [6] and FACADE [7]. Their objective is to support adaptation of the UI at runtime without any modification of its internal code. Adaptation characteristics were integrated as a feature of the operating system so that the user can, for example, move a part of the UI, of a given application, from a window to an other when this application runs. These approaches propose adaptive UI but not adaptable ones (users adapt UI at runtime).

Handling plasticity characteristics, at early stages of the UI development process, was advocated with the emergence of the reference framework of the CAMELEON project [8]. The leading idea is to design a UI once and apply successive suitable transformations to fit the characteristics (interactions techniques, modalities, environment...etc) of various target platforms. Several techniques and toolboxes resulted from this approach. The most significant ones are USIXML [9], COMET [10], WAHID [11] and MultiModel Widgets[12]. USIXML is an XML-based framework, where a UI is specified once and multiple implementations, for various target platforms, may be produced from this description. In the COMET environment [10], a UI is specified at a logical level and multiple rendering technologies can be used to implement it, using a variety of widgets, thanks to a rich toolbox of UI component running in a wide range of platforms. Similar solutions are also provided by WAHID [11] and MultiModel Widgets[12].

The rendering adaptation was also studied. In AMF [13], patterns of interaction techniques are defined in a multi-agent setting. Each pattern is used to adapt the UI input/output according to the interaction mode it describes. Similarly, the UI Module Adaptation approach of [14] allows a user to adapt the GUI to the context of use (platform, environment, user profile) by rearranging the application's user interface. Here, the user is allowed hide and/or to show elements of the interface, which match user interaction preferences and requirements.

More recently, taking into account the *context of use*, has been addressed, particularly the user profile and environment. In [15] a UI is adapted, according to user profiles, stored in an ontology. User profiles are also in the center of nomadic adaptable UI design [16][17], exploited in MAGALLEN [18] to synthesize UI prototypes. In the ubiquitous widgets approach [19], user interactions are captured and transmitted by special components, called IBC (Interactor Business Component) which are used to implement adaptive UI widgets.

This review of existing work shows that there is a need to focus on the interaction between the user and the interface in the plastic UI development process. Moreover, the verification of the plasticity property of UI should be formally and explicitly addressed in the same development process.

B. *Making domain knowledge explicit to handle plasticity*

According to [2], the rapid evolution of IT technologies increased the ubiquity of user interfaces (outside its classical platforms). The diversity of technologies and platforms, equipped with various interaction devices and modalities, is an important factor in the design space of plastic user interfaces [20].

Various approaches in the literature propose to handle this diversity: model-driven approaches, operating system (runtime) approaches and toolboxes approaches[21]. A review of the proposed approaches highlights the need to identify different technologies (interaction devices, rendering technologies, implementation techniques and languages etc.) which equips computing platforms of the market. The most known are interaction patterns used with AMF [22], directory of UI development technologies in COMET environment [10] and user profile-based UI transformation algorithm MAGALLEN [18]. In AMF different interaction techniques are specified as patterns and used in adaptation strategies. The COMET environment proposes to maintain a directory of known rendering technologies. This directory is explored by a COMET (plastic component) to determine which rendering technology is suitable to a given target platform. MAGALLEN proposes to apply a genetic algorithm to implement a user profile-based transformation mechanisms. In these approaches, the design of the UI is seen as an exploration of the design space to find a suitable UI to fit the suited user profile.

Cataloguing devices, interaction modes, platforms, etc. and the relations that hold among them, helps the designer to develop plastic interfaces. The first attempt to model this knowledge is USIXML [9]. A definition of the concepts, manipulated by different models of a UI design process, is provided and modelled thanks to an ontological meta-model. This meta-model drives the generation of a user interface with various rendering target platforms. However, the ontology is terminological one, based on words and terms instead of being a conceptual ontology. We also quote the work of [15], where the same kind of ontology was used to adapt the user interface of an information system, according to the user profile in transportation domain.

Our claim is that user interface adaptation cannot be achieved without the use of a repository, storing all the knowledge about existing interaction techniques, devices and technologies. We believe that conceptual ontologies [23] are well adapted to support such descriptions.

C. *An ontology for plasticity: our view*

In our approach, we advocate the use of conceptual domain ontologies, which rely on the description of concepts characteristics and on ontological reasoning (subsumption computation), in order to determine whether a given concept or an expression of concepts may replace (equivalence, subsumption or entailment) by another concept or expression on concepts. Therefore, our ontology model includes user-tasks expressions and interaction devices (See figure 3). Instances of the concepts (See figure 9) of our ontology are a formalization of task models, describing interaction techniques similar to those modeled by the patterns of AMF [22], or by ICON [24], or in the taxonomy of interaction devices of [25]. The ontology we propose plays a role of a dictionary of interaction techniques [26] [27], representing the various ways to perform various user tasks by various interaction devices. In other words, task models, stored by the instances of our ontology, describe patterns of interactions techniques, used to perform almost known user tasks, in a similar manner with those sketched in [28] (select, copy, text input ...etc).

Finally, note that the definition of an ontology beside of the UI design models provides with a loosely coupled approach, authorizing the evolution of the ontology independently of the models and vice versa.

III. Our approach

Our goal is to verify that two task models, describing two different interactions, are two implementations of the same task with different interaction devices. In this approach, task models are represented by labelled state transition systems (lts) and are compared, using the bi-simulation relationship. If these lts are bi-similar then the task models they represent describe the same user task. Because physical actions of these task models are not the same, due for e.g. to the use of different devices, labels in the corresponding lts may not be the same. At this level, ontology reasoning is triggered to check whether the actions corresponding to these labels have the same semantics. If so, they are rewritten so as the lts share the same set of labels and offer the possibility of a bi-simulation checking. When possible, lts labels are rewritten, thanks to the relations available in the formalizing part of the HCI knowledge domain, related to the user interaction devices and techniques.

Our approach is based on a stepwise methodology, involving the following steps. 1) Build an lts for each task model. 2) Exploit domain ontology relationships to obtain lts sharing the same set of labels. 3) Check bi-simulation for the lts obtained at the previous step. Let us recall some useful definitions.

A. Task modeling with CTT and labelled transition system

CTT [29] notation provides the capability to describe complex user tasks, combining temporal composition operators (inspired from LOTOS process algebra [30]), and atomic tasks (user physical actions on interaction devices). A tree structure is associated to each CTT task model. Figure 1 shows a CTT tree with:

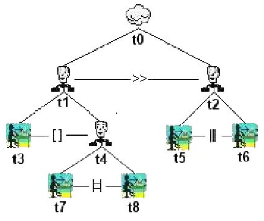

Fig. 1. Example of CTTE task model

- Activation ($\gg$): $t_1 \gg t_2$ for t_1 followed by t_2.
- Choice ([ ]): t_3 [ ] t_4 non-deterministic choice on t_3 or t_4.
- Order independence ($\models|$): $t_7 \models| t_8$ for $(t_7 \gg t_8)[\,](t_8 \gg t_7)$.
- Interleaving (|||): t_5 ||| t_6 parallel activation of t_5 and t_6 .
- Iteration $T^\star$: T is activated zero or more times.

Definition 1: Labelled Transition System. A Labeled Transition System (*lts*) is a structure $\langle S,\ E,\ \longrightarrow\rangle$ where S is a set of states, E is a set of actions (operation labels) and $\longrightarrow\ \subseteq S\times E\times S$ is the transition relation. We write $s \xrightarrow{e} s'$ to express the transition from state s to s' with action e. Internal actions are hidden with the label τ.

In our approach, an *lts* describes the behavior of the interaction between the user and the system, when achieving a task. Physical actions (leaves) of the task model become transition labels and temporal operators are compositions of transitions. Figure 2 shows the automaton, representing the task model of the figure 1.

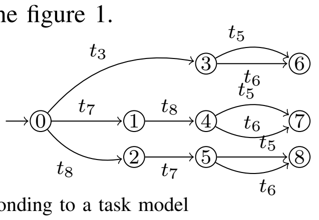

Fig. 2. lts corresponding to a task model

B. The bi-simulation relationship

The notion of *bi-simulation* was initially defined by Milner [31] to compare processes (called observational equivalence). This relation defines equivalence on states of *lts*. Depending on the kind of considered actions (internal τ or observable), two kinds of observational equivalence exist: *strong bi-simulation*, for observable actions, and *weak bi-simulation* which considers both observable and internal τ (stuttering) actions [32].

We consider the definition of bi-simulation for two different lts. Let $lts = \langle S, s_0\ E,\ \longrightarrow\rangle$ and $lts' = \langle S', s'_0\ E,\ \longrightarrow'\rangle$ be two transition systems *with the same set of labels* E and $S \cap S' = \emptyset$.

Definition 2: Strong bi-simulation. Strong bi-simulation relation $\sim\subseteq S \times S'$, is a *bi-simulation equivalence* defined on states. For a given action e, a transition $\xrightarrow{e}$ and two states p_i and q_i, we say that $(p_i, q_i) \in \sim$ if

$$\forall p_i \xrightarrow{e} p_j \in\longrightarrow:\ \exists q_i \xrightarrow{e}{}' q_j \in\longrightarrow' \wedge (p_j, q_j) \in\sim$$
$$\forall q_i \xrightarrow{e}{}' q_j \in\longrightarrow':\ \exists p_i \xrightarrow{e} p_j \in\longrightarrow \wedge (q_j, p_j) \in\sim$$

By extension of this definition, two *lts* are strongly bi-similar, noted $\simeq_{lts}$, if their initial states are bi-similar i.e. $(s_0, s'_0) \in \sim$ (a strong bi-simulation including initial states can be built).

Definition 3: Weak bi-simulation. Weak bi-simulation relationship noted $\approx\ \subseteq S \times S'$, is a bi-simulation equivalence defined on states. For a given action e, a transition $\xrightarrow{e}$ and two states p_i and q_i, we say that $(q_i, q_i) \in\approx$ if

$$\forall p_i \xrightarrow{e} p_j \in\longrightarrow \exists q_i \overset{\tau^\star.e.\tau^{\star\prime}}{\longrightarrow} q_j \in\longrightarrow' \wedge (p_j, q_j) \in\approx$$
$$\forall p_i \xrightarrow{\tau} p_j \in\longrightarrow \exists q_i \longrightarrow^{\tau^\star} q_j \in\longrightarrow' \wedge (p_j, q_j) \in\approx$$
$$\forall q_i \xrightarrow{e}{}' q_j \in\longrightarrow' \exists p_i \overset{\tau^\star.e.\tau^\star}{\longrightarrow} p_j \in\longrightarrow \wedge (q_j, p_j) \in\approx$$
$$\forall q_i \xrightarrow{\tau}{}' q_j \in\longrightarrow' \exists p_i \xrightarrow{\tau^\star} p_j \in\longrightarrow \wedge (q_j, p_j) \in\approx$$

By extension two *lts* are weakly bi-similar, noted $\approxeq_{lts}$, if their initial states are weakly bi-similar i.e. $(s_0, s'_0) \in\approx$ (a weak bi-simulation including initial states can be built).

Bi-simulation relationships are defined on a single set of labels and compare transitions with the same labels. To check plasticity, comparing *lts* with different sets of transition labels is required.

C. A formal model for designing plastic interfaces

The representation of task models by Interactive systems, proposed in our approach, leads to *lts* with different sets of labels, corresponding to interaction actions. Relational bi-simulation relationship, defined below, relaxes the classical definition to handle different sets of labels. This definition introduces a relation, Γ on pairs of labels, used to rewrite different labels.

Let $lts = \langle S, s_0\ E,\ \longrightarrow\rangle$ and $lts' = \langle S', s'_0\ E',\ \longrightarrow'\rangle$ be two transition systems such that $S \cap S' = \emptyset$, $E \nsubseteq E'$ and

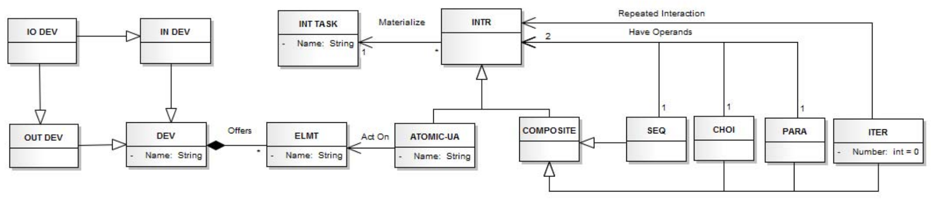

Fig. 3. The ontology model represented as an UML class diagram

$E' \nsubseteq E$. Let A be another set of labels different from the those of $E \cup E'$, in other words $A \cap (E \cup E') = \emptyset$

Definition 4: Bi-directional relation on labels. $\Gamma \subseteq E - E' \times E' - E$ is a relation on labels of two labelled transition systems defined by

$\forall\, \alpha \in E - E', \exists\, \beta \in E' - E$ such that $(\alpha, \beta) \in \Gamma$

$\forall\, \beta \in E' - E, \exists\, \alpha \in E - E'$ such that $(\alpha, \beta) \in \Gamma$

The left and right projection functions $Proj_l$ and $Proj_r$ are associated to Γ.

Informally, Γ links labels belonging to the set of labels that are not in $E \cap E'$. Note that for plasticity checking purpose, this relation will be borrowed from the defined domain ontology.

Definition 5: Rewriting function on labels. Let $\Phi : E \times E' \longrightarrow (A \cup E \cup E' \cup \{\tau\})$ be a function on labels of two labelled transition systems. Φ is defined by

$\forall\, (\alpha, \beta) \in \Gamma \;\; \exists\, \gamma \in A$ such that $\Phi(\alpha, \beta) = \gamma$

The definition of the rewriting function entails four different applications.

1- Substitution. If $\exists\, e \in A$ such that $\Phi(a, b) = e$ then labels a and b are replaced by a new label e in A.

2- Right replacement. If $\exists\, b \in E'$ such that $\Phi(a, b) = a$ then label $b \in E'$ is replaced by a label $a \in E$.

3- Left replacement. If $\exists\, a \in E$ such that $\Phi(a, b) = b$ then label $a \in E$ is replaced by a label $b \in E'$.

4- Hiding. $\Phi(a, b) = \tau$ denotes the case of a pair of labels that should be hidden on both labelled transition systems after rewriting.

Remark. When an ontology is used, substitution means that a subsuming concept is available and left (resp. right) replacement means that the left (resp. right) label is subsumed by the right (resp. left) one. Hiding means that the label is not relevant for plasticity. *If* Γ *is empty, then no plasticity is possible.*

Definition 6: Transforming labelled transition systems. $lts = \langle S, s_0\ E, \longrightarrow\rangle$ and $lts' = \langle S', s'_0\ E, \longrightarrow'\rangle$ are respectively rewritten to $lts^{\top} = \langle S^{\top}, s_0^{\top}\ E^{\top}, \longrightarrow^{\top}\rangle$ and $lts^{\top'} = \langle S^{\top'}, s_0^{\top'}\ E^{\top'}, \longrightarrow^{\top'}\rangle$ according to the label relation Γ and to the rewriting function on labels Φ as follows .

-$S^{\top} = S$ and $S^{\top'} = S'$ i.e. same sets of states.

- $s_0^{\top} = s_0$ and $s_0^{\top'} = s'_0$ i.e. same initial states.

- $E^{\top} = (E - Proj_l(\Gamma)) \cup A \cup \{\tau\}$ and $E^{\top'} = (E' - Proj_r(\Gamma)) \cup A \cup \{\tau\}$ sets of labels enriched with the rewritten labels thanks to the Φ rewriting function.

-transition relations are redefined on the new labels $\longrightarrow^{\top} \subseteq S^{\top} \times E^{\top} \times S^{\top}$ and $\longrightarrow^{\top'} \subseteq S^{\top'} \times E^{\top'} \times S^{\top'}$ where

$$\begin{aligned}
\longrightarrow^{\top} &= \{s \xrightarrow{e} t \in \longrightarrow \mid \forall e' \in E'.\ (e, e') \notin \Gamma\} \\
&- \{s \xrightarrow{e} t \in \longrightarrow \mid \forall e' \in E'.\ (e, e') \in \Gamma\} \\
&\cup \{s \xrightarrow{a} t \mid \exists (e, e') \in \Gamma \wedge \Phi(e, e') = a\} \\
\longrightarrow^{\top'} &= \{s' \xrightarrow{e'} t' \in \longrightarrow' \mid \forall e \in E.\ (e, e') \notin \Gamma\} \\
&- \{s' \xrightarrow{e'}{}' t' \in \longrightarrow' \mid \forall e \in E.\ (e, e') \in \Gamma\} \\
&\cup \{s' \xrightarrow{a}{}' t \mid \exists (e, e') \in \Gamma \wedge \Phi(e, e') = a\}
\end{aligned}$$

The defined $lts^{\top}$ and $lts^{\top'}$ are labelled transition systems with the same set of labels, since $E^{\top} = E^{\top'}$.

Definition 7: Relational weak bi-simulation relationship on lts. Let $\langle lts, lts', \Gamma, \Phi\rangle$ be a structure where

-lts and lts' are two labelled transition systems such that $S \cap S' = \emptyset$, $E \nsubseteq E'$ and $E' \nsubseteq E$,

- $\Gamma \subseteq E \times E'$ is a relationship on labels according to definition 4,

- Φ is a label rewriting function according to definition 5.

Then, $\approxeq_{lts}^{\Gamma,\Phi} \subseteq LTS \times LTS$ is relational weak bi-simulation relationship on labelled transition systems if there exists a weak bi-simulation relationship on labelled transition systems between the transformed lts. We write

$$(lts, lts') \in \approxeq_{lts}^{\Gamma,\Phi} \Longleftrightarrow (lts^{\top}, lts^{\top'}) \in \approxeq_{lts}$$

It becomes possible to compare labelled transition systems with different sets of labels.

Definition 8: Plasticity property. For two user interfaces UI_1 and UI_2, modeled by two task models, corresponding to the labelled transition systems lts_1, lts_2; and an ontology Ont, defining the relation Γ_{Ont}; and a rewriting function Φ_{Ont}; we say that UI_1 and UI_2 satisfy plasticity property if $lts_1 \approxeq_{lts}^{\Gamma_{Ont},\Phi_{Ont}} lts_2$. Note that if $\Gamma = \emptyset \wedge E1 \neq E2$ the plasticity property does not hold.

D. An ontology of interaction

A domain ontology of both tasks and interaction devices is defined to supply *Relational bi-simulation* with the relation Γ and the rewriting function Φ on labels. The different actions performed on an interactive system, together with their associated devices, can be categorized within an ontology. Classes and properties related to interaction devices and modes are categorized in a subsumption relationship. Figure 3 shows a class diagram, representing the core concepts of the obtained ontology. It defines hierarchical categories of interactive actions and/or devices. The main concepts of this ontology are the *interaction device* (DEV), the *interactive task* (INTTASK) and the *user interaction* (INTR). *Interaction device (DEV)* describes the various categories of devices. It is mainly based on most known taxonomies of interaction devices, published in the literature [26], [25], [33] and [34]. The concept *Interactive task (INTTASK)* refers to the abstract interaction, as defined in [28] -examples of instances of this class are *select, copy, text input* etc. *User interaction (INTR)* class represents the patterns of interactions techniques as defined in [27], [22] and [24]. It can be either a basic interaction, corresponding to a *user action* (ATOMIC-UA), or a composite action, defining the behavior of the user and the machine during the interaction in terms of *user actions*. The actions composition operators are the sequence (SEQ), concurrent (PARA), choice (CHOI) and iteration (ITER). Each interaction materializes an alternative way to perform an *interactive task (INTTASK)* with a set of *user actions*, allowed by *interaction devices (DEV)* available within a computing platform.

Among the different kinds of ontological relations between *user actions*, offered by our ontology, the *equivalence* and the *subsumption (is_a)* relationships are exploited by our approach. The former means that two user interactions *INTR* are equivalent when they materialize the same interactive task *INT TASK*. The latter describes the situation where an atomic task can be performed either by an atomic action *ATOMIC_UA* or by a composite task *COMPOSITE*. Particularly, the relation linking the concept *INT TASK* and the concept *INTR* is exploited when rewriting labels of the lts in the following manner:

• A label, derived from equivalent *ATOMIC_UA*, is replaced by the label, derived from the *INTR* they perform (*materialize*). Equivalence is a substitution.

• A label, derived from *INT TASK*, replaces two sets of equivalent *COMPOSITE* interactions (both realize the same *INT TASK*)

• A label, derived from *INTR*, can replace a set of labels derived from an *ATOMIC_UA* or *COMPOSITE* interactive task if it is subsumed by a single *ATOMIC_UA* that realizes the same *INT TASK*. Subsumption is a left or right replacement.

Finally, as for classical ontology engineering, the ontology, defined above, shall be consensual and agreed by the user interface developers' community. This aspect is out of scope of our paper and the presented ontology can be subject to evolution or amendment without impacting the defined approach.

E. A Methodology for checking user interface plasticity

The main goal in our proposed approach is to check that a pair of different interactive systems allows a user to perform the same tasks, using different interactive techniques with available interactive devices and modes. Considered interactive systems are formalized as labelled transition systems, and then a relational bi-simulation on these systems is checked, provided that a relation on labels of their corresponding labelled transition systems is available -using our ontology. Informally, our approach states that a transition system is said plastic if it can be at least implemented by two interactive systems, which are linked by a relational bi-simulation relationship, according to a given relation on labels. Our stepwise methodology, to check that a pair of interactive systems $(Syst_{source}, Syst_{target})$, consists of the following steps:

1- *Design*. Formalize interactive systems $(Syst_{source}, Syst_{target})$ as a pair $(lts_{source}, lts_{target})$ of labelled transition systems.

2- *Irrelevant action identification*. In the obtained *lts*, at the previous step, identify the internal actions that are not relevant for the interaction. The labels, corresponding to these actions in lts_{source} and/or lts_{target}, are set to τ.

3- *Rewriting*. Identify the pairs of labels that are different in lts_{source} and lts_{target}, obtained at the previous step. Then, rewrite these pairs of labels, using the relation on labels of the *lts*, defined in the interaction ontology. At this step, the two lts_{source} and lts_{target} have the same labels.

4- *Checking*. Check classical weak bi-simulation between *LTS* lts_{source} and lts_{target}, obtained at the previous step. If they are bi-similar, then we can assert that they encode the same task model.

IV. Application to a case study

In order to illustrate the previously described formal method for checking UI plasticity, we consider the case of checking if a web application, initially designed to send SMS on a personal computer (PC), runs on a Smartphone as well. Below, we show how two task models, describing two user interfaces, concretizing the same task on these two concrete different platforms (PC and Smartphone).

A. Tasks description

Figure III-E shows the CTT task model, describing how the application UI allows a user to send an SMS. The user opens a session to access his own space (subtask TA). Then, he writes a message and sends it (subtask TB). Finally, he logs out from his own space (subtask TC). The leaves of the defined task model are actions that a user must carry out to achieve each subtask. To login, a user gives his identifier (T_1) and password (T_2) in any order and submits them (T_3). Then, he triggers the message editor (T_4), edits his message (T_5) by entering the text of the message (T_8) and the phone number of the recipient (T_7) (in any order). Then, he decides whether he sends the entered message (T_6) or saves it (T_{10}) or cancels it (T_{11}). To send the message, the user must request the system (T_{12}) and confirm the action (T_{13}). If he does not confirm sending the message (T_{14}), he chooses to save it (T_{15}) or to cancel the whole operation (T_{16}). Finally, the user closes the session (T_{17}).

B. Design

In order to keep this paper in a reasonable length, we only describe the subtask *EditMsg* (T_5). We show how our formal approach applies to establish the similitude (equivalence) of this subtask, when deployed on a PC or on a Smartphone. The Task model of figure 5 describes how to carry out the task (T_5) on the PC platform.

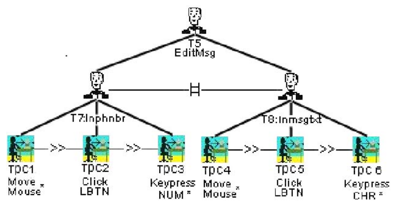

Fig. 5. The *EditMsg* subtask (T_5) deployed on a PC

The model of figure 6 shows how the same task is achieved on a Smartphone. As a consequence, two interactive systems $(Syst_{source}, Syst_{target})$, corresponding respectively to the PC and to the Smartphone platforms, are described.

From the task models of figures 5 and 6, a pair (lts_{Pc}, lts_{Ph}) of *lts*, formalizing the behavior of $(Syst_{source}, Syst_{target})$, are obtained. As mentioned above, CTT models are formalized as LOTOS process algebra expressions and thus an *lts* can be associated to each CTT expression.

The *lts* lts_{Pc} and lts_{Ph} of figures 7 and 8 correspond to $Syst_{source}$ and $Syst_{target}$, respectively. They describe the interactive system, running on a PC and on a Smartphone respectively.

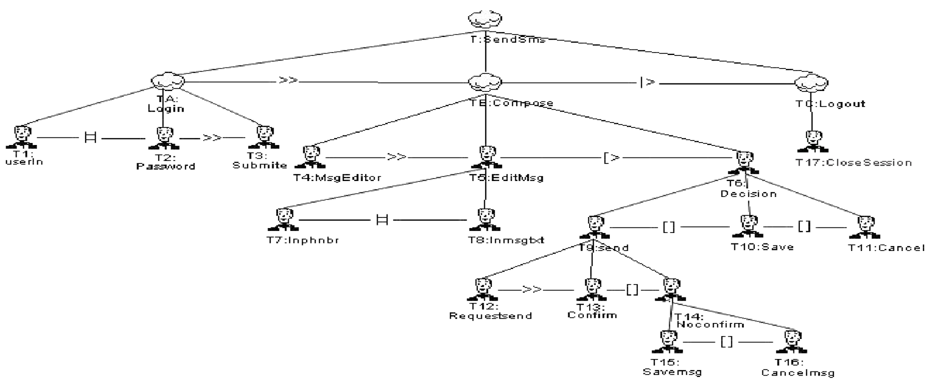

Fig. 4. Send SMS abstract Task Model

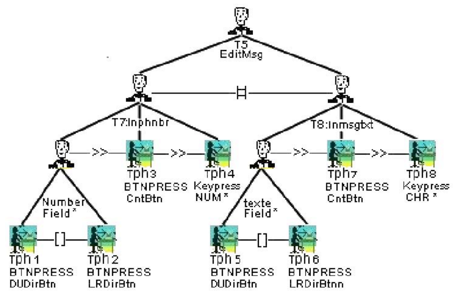

Fig. 6. The *EditMsg* subtask (T_5) deployed on a Smartphone

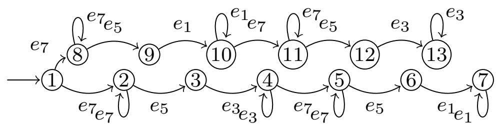

Fig. 7. lts on PC platform

TABLE I. USER ACTIONS AND CORRESPONDING LABELS

CTT's atomic actions	Signification	*lts*'s Labels
KeyPress CHR	Press the keyboard character key	e_1
KeyPress NUM	Press the keyboard digital key	e_3
Click LBTN	Click the mouse left button	e_5
BtnPress DUDirBtn	Press keypad Down/Up direction button	e_6
Move Mouse	Move the mouse	e_7
BtnPress LRDirBtn	Press keypad Left/Right direction button	e_8
BtnPress CentBtn	Press keypad central button	e_{10}

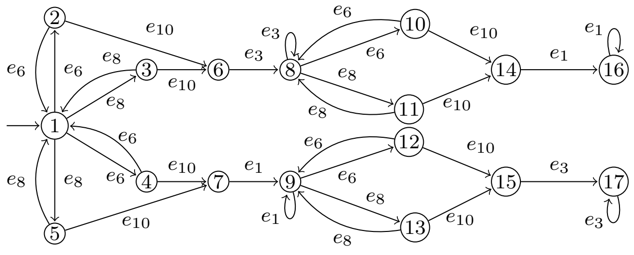

Fig. 8. lts on Smartphone platform

Table I gives the correspondence between actions of task models and the corresponding labels in the *lts*.

C. Identification of irrelevant actions

The labels, corresponding to irrelevant actions in the task model with respect to the plasticity, are identified from the ontology. In this case study, the iterated user actions, corresponding to moving the mouse in the screen on the PC {*MoveMouse*}, or moving buttons on the Smartphone {*BTNPressDUDirBtn, BTNPressLRDirBtn*}, are considered as irrelevant. Only one iteration is considered in this case and the other iterations are set to τ for their corresponding labels (e_7, e_6, e_8) in lts_{Pc} and/or lts_{Ph}. Note that these actions could have been kept as they are if one may consider the number of times buttons are pressed, for example.

D. Rewriting

The next step consists in rewriting the different *lts*, involved in the task models, corresponding to (lts_{Pc}, lts_{Ph}), in order to produce new *lts* that share the same set of labels. Let E_{Pc} and E_{Ph} be the set of labels, corresponding to lts_{Pc} and lts_{Ph}, respectively. $LabDiff$ is the set of labels of lts_{Pc} and lts_{Ph}, which are different.

$$\begin{aligned} LabDiff &= (E_{Pc} \cup E_{Ph}) - (E_{Pc} \cap E_{Ph}) \\ &= \{e_5, e_6, e_7, e_8, e_9, e_{10}\} \end{aligned}$$

The set of labels E_s^r and E_t^r to be rewritten for lts_{Pc}, and lts_{Ph}, respectively, are defined below.

$$\begin{aligned} E_{Pc}^r &= E_{Pc} \cap LabDiff & E_{Ph}^r &= E_{Ph} \cap LabDiff \\ &= \{e_5, e_7\} & &= \{e_6, e_8, e_{10}\} \end{aligned}$$

TABLE II. LABEL REWRITING TABLE

***lts*'s Labels**	**Substitution action**	**ϕ Application**
(e_5, e_{10})	GO	$\phi(e_5, e_{10}) = g$
(e_7, e_6)	MoveCursor	$\phi(e_7, e_6) = m$
(e_7, e_8)	MoveCursor	$\phi(e_7, e_8) = m$

At this level, according to the defined methodology, the rewriting function can be applied. This function is defined in table II. Figure 9 shows an excerpt of the ontology instances, used for this case study. Labels (g, m) appear after rewriting. The label g, corresponding to the *interactive task (INT TASK) GO*, replaces the labels corresponding to *User Actions* $(ATOMIC\_UA)$, *Click* that *acts on* LBTN, and *BtnPress* that *acts on CentBtn*; because it represents the effect of the two actions. The label *m*, corresponding to the *interactive task (INT TASK) MoveCursor*, replaces the labels corresponding to user actions ($ATOMIC\_UA$), *Move* that *acts on Mouse*, *BtnPress* that *acts on DUDirBtn*, and *BtnPress* that *acts on LRDirBtn*.

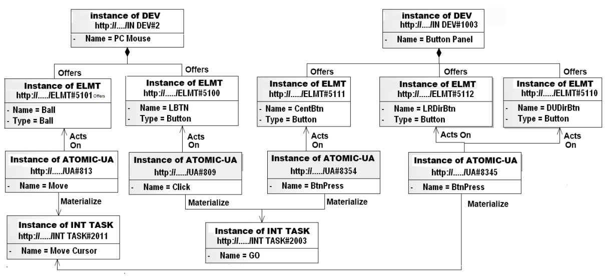

Fig. 9. Instances of our ontology concepts used to rewrite labels

E. Checking

Once the previous steps are achieved, the labelled transition systems lts_{Pctr} of figure 10 and the lts_{Phtr} of figure 11 are obtained for both the PC and the Smartphone, respectively. They share the same set of labels $\{e_1, e_3, m, g, \tau\}$.

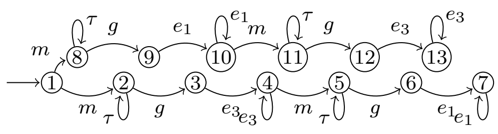

Fig. 10. lts on PC platform after rewriting labels

It becomes possible to verify that the obtained *lts* are bi-similar by checking the *observational equivalence* relationship between them, i.e. by checking the weak bi-simulation relationship, previously defined.

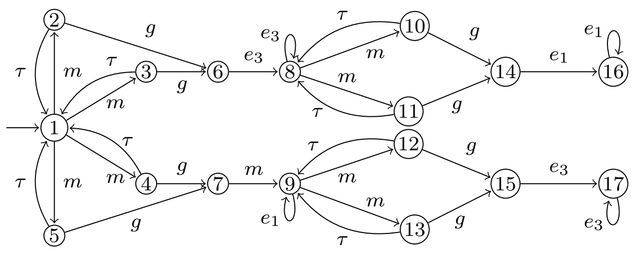

Fig. 11. lts on Smartphone platform after rewriting labels

The final result shows that lts_{Pctr} and lts_{Phtr} are weakly bi-similar. Therefore, we can conclude that the task models, described on a PC and on a Smartphone, can substitute each other, ensuring thus the plasticity property.

F. Tool support

The $CADP^{©}$ (Construction and Analysis of Distributed Processes) model checker [35] has been set up to model tasks and check bi-simulation. This toolbox provides resources for compiling and checking Lotos [30] programs. In our approach, the *lts* are first, formalized in LotosNt [35] (see figure 12), a simplified version of LOTOS. They are transformed into LOTOS programs. Then, an internal representation, by labelled transition system (automaton) in BCG format, is generated for each LOTOS program. Finally, the bi-simulation of the two BCG automata is checked, thanks to the BISIMULATOR module, using *observational equivalence* relationship option.

```
module ltspct is
process main [m,g,e1,i,e3 :any] () is
   select
      m; brch1[m,g,e1,e3,i] []  m; brch2[i,m,g,e1,e3]
   end select
end process -- main
process brch1 [m,g,e1,i,e3:any] () is
   select
      i; brch1[m,g,e1,e3,i] [] g; e1; brch11[m,g,e1,e3,i]
   end select
end process -- brch1: branche1
......
process brch23[e1:any] ()is
   e1; brch23[e1]
end process
end module -- ltspct
```

Fig. 12. A section of LOTOS NT code corresponding to lts_{Pctr}

V. Conclusion

This paper presented a formal modeling approach to check the plasticity property of user interfaces. The objective is to allow UI developers check whether different user interfaces, running on different computing platforms, with different interaction techniques, and various available interaction devices, support the same user tasks. The approach relies on the formalization of task models by *lts*, where states record the UI changes and transitions are labelled with user actions. They offer a powerful model to describe behavioral aspects of task models. Our idea, to handle plasticity of user interfaces, comes from the fact that, once task models are formalized as *lts*, we observe that different paths with different actions of two task models may lead to *equivalent* states. This notion of equivalence is hard to establish because the actions, labeling the transitions, are not *exactly* the same ones. There is a need to express equivalence relations on such actions when such a relation holds. Therefore, one needs to make explicit the knowledge that some interaction devices, modes or actions have the same effect. This information represents an axiomatization of the interaction domain. We have chosen to model such a knowledge with domain ontologies, that are well adapted for the description of domain knowledge independently of any context of use. The proposed approach is based on two formal verification techniques. First, it exploits the bi-simulation relationship to compare behaviors, expressed by *lts*. Second, the computation of the subsumption and equivalence relationships, within an ontology provided with equivalent or subsuming instances, that can be used to safely rewrite transition labels. The interest of the proposed approach is its loose coupling between the tasks models -the user interface- and the ontology. Indeed, both may evolve independently.

Moreover, this approach can be used for handling plasticity at both design and runtime. On the one hand, checking bi-simulation overall labelled transitions systems, ensures that a global task model can substitute another one. On the other hand, the fact that the bi-simulation relationship is defined on pairs of states ensures that two bi-similar states are equivalent. Thus, it becomes possible to switch from one state of an *lts* to its bi-similar state, of another *lts*, at runtime. A case study of a web application, sending an SMS from a Pc or from a Smartphone, showed how our approach works. It illustrates two equivalent task models modulo an interaction domain ontology.

Finally, this work opens several research perspectives. First, as for classical ontology engineering, the ontology used in this paper shall be consensual and agreed by the UI developers' community. Second, the definition of the rewriting function should be automated. We are currently investigating how this function can be encoded within rewriting systems like Maude. Third, more complex applications should be addressed in order to show how this approach scales up to other interactive systems. Finally, we believe that the provided relational bi-simulation relationship opens research paths for studying adaptive systems in general.

References

[1] D. Thevenin and J. Coutaz, "Plasticity of user interfaces: Framework and research agenda," in *Proceedings of INTERACT*, vol. 99, 1999, pp. 110–117.

[2] J. Coutaz and G. Calvary, "HCI and Software Engineering for User Interface Plasticity," in *HCI Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, 3rd Edition*, J. A. Jacko, Ed. CRC Press, 2012, pp. 1195–1220.

[3] J. A. Johnson, B. A. Nardi, C. L. Zarmer, and J. R. Miller, "Ace: building interactive graphical applications," *Communications of the ACM*, vol. 36, no. 4, pp. 40–55, 1993.

[4] S. Kawai, H. Aida, and T. Saito, "Designing interface toolkit with dynamic selectable modality," in *Proceedings of the 2nd annual ACM conference on Assistive technologies*. ACM, 1996, pp. 72–79.

[5] M. Crease, S. Brewster, and P. Gray, "Caring, sharing widgets: a toolkit of sensitive widgets," in *People and Computers Usability or Else!* Springer, 2000, pp. 257–270.

[6] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses: the see-through interface," in *Proceedings of the 20th annual conference on CGIT*. ACM, 1993, pp. 73–80.

[7] W. Stuerzlinger, O. Chapuis, D. Phillips, and N. Roussel, "User interface faades: towards fully adaptable user interfaces," in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, ser. UIST '06. New York, NY, USA: ACM, 2006, pp. 309–318.

[8] G. Calvary, J. Coutaz, L. Bouillon, M. Florins, Q. Limbourg, L. Marucci, F. Paternò, C. Santoro, N. Souchon, D. Thevenin *et al.*, "The cameleon reference framework," *Deliverable D1*, vol. 1, 2002.

[9] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero, "Usixml: A language supporting multi-path development of user interfaces," *Engineering HCI and Interactive Systems*, pp. 134–135, 2005.

[10] A. Demeure, G. Calvary, and K. Coninx, "Comet(s), a software architecture style and an interactors toolkit for plastic user interfaces," in *Interactive Systems. Design, Specification, and Verification*, ser. LNCS, T. Graham and P. Palanque, Eds. Springer Berlin Heidelberg, 2008, vol. 5136, pp. 225–237.

[11] B. Jabarin and T. Graham, "Architectures for widget-level plasticity," *Interactive Systems. Design, Specification, and Verification*, pp. 451–460, 2003.

[12] A. Stanciulescu, *Methodology for Developing Multimodal User Interfaces of Information Systems (a)*. Presses univ. de Louvain, 2008, vol. 556.

[13] K. Samaan and F. Tarpin-Bernard, "The amf architecture in a multiple user interface generation process," in *Developing User Interfaces with XML, AVI'2004 Workshop*, 2004.

[14] A.-M. Dery-Pinna, J. Fierstone, and E. Picard, "Component model and programming: a first step to manage human computer interaction adaptation," in *Human-Computer Interaction with Mobile Devices and Services*. Springer, 2003, pp. 456–460.

[15] K. M. De Oliveira, F. Bacha, H. Mnasser, and M. Abed, "Transportation ontology definition and application for the content personalization of user interfaces," *Expert Systems with Applications*, vol. 40, no. 8, pp. 3145–3159, 2013.

[16] J. Sonnenberg, "Service and user interface transfer from nomadic devices to car infotainment systems," in *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM, 2010, pp. 162–165.

[17] W. Dees, "Usability of nomadic user interfaces," in *Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments*, ser. LNCS, J. Jacko, Ed. Springer Berlin Heidelberg, 2011, vol. 6763, pp. 195–204.

[18] D. Masson, A. Demeure, and G. Calvary, "Examples galleries generated by interactive genetic algorithms," in *Procedings of 2nd Conf. on Creativity and Innovation in Design*. ACM, 2011, pp. 61–71.

[19] D. Pierre, D. Marc, and R. Philippe, "Ubiquitous widgets: Designing interactions architecture for adaptive mobile applications," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2013, pp. 331–336.

[20] J. Vanderdonckt, D. Grolaux, P. Van Roy, Q. Limbourg, B. Macq, and B. Michel, "A design space for context-sensitive user interfaces," *Proceedings of IASSE*, 2005.

[21] A. Demeure, G. Calvary, J. Coutaz, and J. Vanderdonckt, "The comets inspector: Towards run time plasticity control based on a semantic network," *Task Models and Diagrams for UI Design*, pp. 324–338, 2007.

[22] K. Samaan and F. Tarpin-Bernard, "Task models and interaction models in a multiple user interfaces generation process," in *Proceedings of the 3rd annual conference on Task models and diagrams*. ACM, 2004, pp. 137–144.

[23] S. Jean, G. Pierra, and Y. Ait-Ameur, "Domain Ontologies: A Database-Oriented Analysis," in *Web Information Systems and Technologies, International Conferences, WEBIST 2005 and WEBIST 2006. Revised Selected Papers*, ser. LNBIP. Springer Berlin Heidelberg, 2007, pp. 238–254.

[24] P. Dragicevic and J.-D. Fekete, "Support for input adaptability in the icon toolkit," in *Proceedings of the 6th international conference on Multimodal interfaces*. ACM, 2004, pp. 212–219.

[25] W. Buxton, "A three-state model of graphical input," in *Human-computer interaction-INTERACT*, vol. 90. Citeseer, 1990, pp. 449–456.

[26] S. Berti, F. Paternò, and C. Santoro, "A taxonomy for migratory user interfaces," in *Interactive Systems. Design, Specification, and Verification*. Springer, 2006, pp. 149–160.

[27] D. Navarre, P. Palanque, and S. Basnyat, "A formal approach for user interaction reconfiguration of safety critical interactive systems," in *Computer Safety, Reliability, and Security*. Springer, 2008, pp. 373–386.

[28] L. L. Constantine, "Canonical abstract prototypes for abstract visual and interaction design," in *Interactive Systems. Design, Specification, and Verification*. Springer, 2003, pp. 1–15.

[29] F. Paternò, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models," in *Proceedings of the IFIP TC13 Interantional Conference on HCI*, vol. 96, 1997, pp. 362–369.

[30] I. Lotos, "A formal description technique based on the temporal ordering of observational behaviour," *ISO/IEC JTC 1/SC 7, Geneva*, 1988.

[31] R. Milner, *A Calculus of Communicating Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982.

[32] ——, *Communication and concurrency*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[33] S. Card, J. D. Mackinlay, and G. Robertson, "The design space of input devices," in *Proceedings of ACM Conference on Human Factors in Computing Systems, Multi-Media*, ser. CHI90, 1990, pp. 117–124.

[34] D. Frohlich, "The design space of interfaces," in *Multimedia Systems, Interaction and Applications, 1st Eurographics Workshop*. Springer Verlag, 1991.

[35] CADP-Team, "Cadp web site http://cadp.inria.fr," webpage, 2013.