Front. Comput. Sci.
DOI 10.1007/s11704-016-5460-3

RESEARCH ARTICLE

A formal model for plastic human computer interfaces

Abdelkrim CHEBIEB (✉)[1], Yamine AIT AMEUR (✉)[2]

1 Computer Science School for Engineers (ESI), Algiers 16270, Algeria
2 IRIT-INPT-ENSEEIHT, Toulouse 31071, France

Abstract The considerable and significant progress achieved in the design and development of new interaction devices between man and machine has enabled the emergence of various powerful and efficient input and/or output devices. Each of these new devices brings specific interaction modes. With the emergence of these devices, new interaction techniques and modes arise and new interaction capabilities are offered. New user interfaces need to be designed or former ones need to evolve. The design of so called plastic user interfaces contributes to handling such evolutions. The key requirement for the design of such a user interface is that the new obtained user interface shall be adapted to the application and have, at least, the same behavior as the previous (adapted) one. This paper proposes to address the problem of user interface evolution due to the introduction of new interaction devices and/or new interaction modes. More, precisely, we are interested by the study of the design process of a user interface resulting from the evolution of a former user interface due to the introduction of new devices and/or new interaction capabilities. We consider that interface behaviors are described by labelled transition systems and comparison between user interfaces is handled by an extended definition of the bi-simulation relationship to compare user interface behaviors when interaction modes are replaced by new ones.

Keywords formal modeling and verification, ontology based modeling, plastic user interfaces, adaptive systems

Received November 3, 2015; accepted March 23, 2016

E-mail: k_chebieb@esi.dz; yamine@enseeiht.fr

1 Introduction

A user interface is often designed, after some refinement steps, for predefined interaction devices and platforms. Each device, platform and environment is characterized by its own interaction modes. When a user interface (UI) is designed to run on several target platforms1) and to support different interaction modes and/or devices, it can be considered as satisfying the plasticity property (also qualified to be a plastic UI). Indeed, a plastic UI shall be able to switch, statically (at design time) or dynamically (at runtime), from a given platform to another. The target platform may support (exactly, less or more) interaction capabilities than the original one corresponding to (equivalent, degraded or upgraded) user interface.

With the emergence of new devices, new interaction techniques and modes arise. Indeed, when such devices are deployed to interact with hardware controllers, games, critical applications like medicine or aircraft cockpits, classical software applications, etc., new interaction capabilities are enabled. Therefore, either new user interfaces need to be designed and verified for the obtained system or the former user interface needs to evolve. In other words, some of these user interfaces result from the evolution of the former user interface due to the introduction and/or to the substitution of one or more devices by other ones. In some cases, other user interfaces may require to build a completely new user interface taking into account the introduced new devices including their new interaction modes.

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

As a consequence, the obtained user interface (being either a new one or the evolution of a former one) requires to be (partly or fully) re-designed, re-verified and re-validated although this new user interface (to be defined) still interacts with the same application. The key requirement for the design of such a user interface is that the new obtained user interface shall be adapted to the application and have, at least, the same behavior as the previous one.

The previously identified requirement advocates for the design of so called plastic user interfaces. In this case, the design relies on the concept of plasticity and plastic user interfaces [1]. Plasticity is an important property to ensure the safety and usability of interactive systems which is one of ISO/IEC 9126-1 usability quality of service criteria [2]. It aims at supporting user interface adaptation to several running situations by providing another design model of the whole or part of the user interface. In a dynamic setting, this feature is particularly useful to pursue interacting with the system even if a failing situation occurs [3].

The description of the behavior of user interfaces is a major concern in user interface engineering areas. Several approaches, notations, techniques, processes and methods have been proposed in the literature. Compared to classical software engineering, the design of user interfaces pays a lot of attention to the usability of the designed interface. One of the techniques allowing a designer to handle this usability characteristic in the behavior description is user task specification and analysis. Indeed, a set of user tasks is defined beside or within the user interface specification in order to describe expected and/or unexpected user interface behaviors. Tasks are defined by different actors involved in the description of the user interface (e.g., ergonomists, psychologists, users corresponding to specific profiles like pilots for cockpits interfaces, etc.). Defined user tasks contribute to the verification and validation of the user interface, they define use cases and scenarios. Validation and verification activities consist in checking that the defined user tasks are supported or not supported by the designed user interface. This checking is ensured by any validation and/or verification technique like testing, simulation, experimentation, formal proofs, model checking, etc.

This paper proposes to address the problem of user interface evolution due to the introduction of new interaction devices and/or new interaction modes. More, precisely, we are interested by the study of the design process of a user interface resulting from the evolution of a former user interface due to the introduction of new devices and/or new interaction capabilities. This paper claims to handle the plasticity characteristic of user interfaces by answering to the question: *does a target user interface U_T resulting from the evolution (by introducing new interaction devices or interaction modes) of a source user interface U_S behave as U_S?* To provide with answers to this question, one should be able *to formally compare the behaviors* of each of the considered user interfaces U_S and U_T.

In order to set up our proposal for handling plastic interfaces and checking the capability of a user interface to be replaced by another one, we consider that

- user interfaces are viewed as task models, and formally described as state transitions systems,
- devices and associated interaction modes are formally represented within a knowledge model carried out by a domain ontology,
- and finally, the problem of interfaces behaviors comparison is handled using the classical techniques for comparing state transitions systems. A revisited definition of the classical bi-simulation relationship is provided.

This paper is structured as follows. Section 2 addresses the design of human centered computer interfaces, it gives an overview of the different techniques developed to define user tasks models. Section 3 focuses on the concept of user interface plasticity. It reviews the basic definitions and surveys the state of the art in the design of plastic user interfaces. It also shows how devices and interaction modes can be modeled as an explicit knowledge domain, i.e., an ontology. In Section 4, the core principles of the proposed approach are presented. Then, Sections 5 and 6 revisit the definition of the bi-simulation relationship needed to compare user task models. The whole formal model for verifying plastic user interfaces and the plasticity property is presented in Section 7 where the different steps leading to analyze formally plastic user interfaces are composed into a sequence of methodological steps. Section 8 is devoted to the development of our approach on two illustrative case studies. The use of a model checker for formal verification of plastic user interfaces is described in this section as well. Finally, Section 9 concludes this work and gives some future research directions.

2 Design of human computer interfaces: task modelling

During the user interfaces specification, design, validation and experimentation processes, task modeling allows user interfaces developers to capture usability characteristics through the definition of scenarios of use. These processes heavily rely on the definition of user tasks to be supported by the user interface under design. The described tasks shall be handled by the user interface whatever are the environment and the platform where this user interface is set up. This requirement defines the notion of *abstract task* to be handled by every user interfaces designed to interact with the considered system. The usability of the user interface is checked by asserting that the defined user tasks are handled by the interface. In case of evolution of the user interface (e.g., new interaction device or mode), the defined tasks shall be checked again to ensure interface adaptation. Therefore, as mentioned above, task models suit for plasticity of user interfaces checking.

Note that other notations and modeling languages are available for design and specification of user interfaces. In this paper, we focus on task models. In the remainder of this section, we review the main research work related to formal user tasks modeling and give an overview of the concur task tree (CTT) user task modeling language defining a process algebra and used in our approach for tasks description.

2.1 User task modeling

Task modeling, initially used for requirement analysis and knowledge specification, is the starting point of user interface design and development best practices [4]. It is the backbone of user interface development process. Indeed, task modeling has been used in several situations : to derive several user interfaces like in TERESA [5], to determine facets in agent multi facets (AMF) [6], to determine task view point in multi-path development [7], etc. Their common objective is to develop usable and useful systems [2].

The study of task modeling languages and techniques has drawn user interaction researcher's attention since the 80's. Tasks may be described at different levels:

- an *abstract level* for the description of the task actions to be performed through the user interface and at
- a *concrete level* depending on the available devices and interaction modes within a given platform or environment.

The expressive power of task modeling languages resides in their capability to describe different levels of the task: abstract level, syntax (or structural) level and concrete (or keystroke) level. Indeed, according to [8], in general a task model is composed of three layers. First, a top layer describes the task to be achieved at the abstract level. Second, a mid layer models the dialogue and indicates the user actions (cognitive decisions), system action and interactive actions (shared between a user and a machine or a system). Finally, the third (bottom) layer is a concretization. It models the physical actions (keystroke interactions) needed to perform the described task depending on available interaction devices of the platform where the system is expected to run.

Several techniques, notations and editing tools are dedicated to process and analyze task models. The most known in the literature are *UAN* [9, 10], *XUAN, Xuan*, *HTA* [11], *CTT* [12,13], *MAD* [14] and *MAD** [15] and its tool *KMADE* [16].

2.2 Formal modeling and verification of user tasks

The use of formal methods for the validation of user interfaces and particularly task models has been studied by several authors. Various techniques, tools and models have been proposed as a solution to support human centered design such as task modeling [12], task achieving verification [17] multimodal user interface [18], and user driven design [19]. Tasks descriptions are formally modeled by labelled transition systems (*lts*). This representation makes it possible to target several formal verification techniques. Indeed, to encode task models as labelled transition systems, Petri Nets [20], process algebra, based on the LOTOS with CTT and CTTE [12, 21], state based methods with B and Event-B [22, 23] or Z [24], model checking and temporal logics by [25–27], etc., are some of the approaches that have been proposed in the literature so far. These approaches show the attention carried out by research to the problem of formal modeling and verification of task models for both WIMP or post-WIMP user interfaces.

2.3 The concur task tree (CTT) notation

CTT is a notation for task model specification used to design interactive applications. It provides a designer with a notation to describe tasks expressions combining temporal operators of a process algebra *à la CCS (Calculus of Communicating Systems* [28]) and atomic tasks (user physical or keystroke actions on interaction devices). A CTT task model describes a hierarchy of tasks represented by a tree-like structure, where each node represents a composition operator and each leaf is an atomic task. It requires identification of temporal relationships between sub-tasks of the same tree level. These operators are borrowed from the LOTOS process algebra [29]. The available CTT composition operators describe activation ($T_i \gg T_j$), choice (T_i [ ] T_j) order independence ($T_i \models\mid T_j$), interleaving $T_i \mid\mid\mid T_j$, parallel tasks ($T_i \mid\mid T_j$) and iteration

(T^*).

Figure 1 corresponds to the decomposition tree describing the $T_3[\,](T_7 \models\!| T_8) \gg (T_5 ||| T_6)$ task expression.

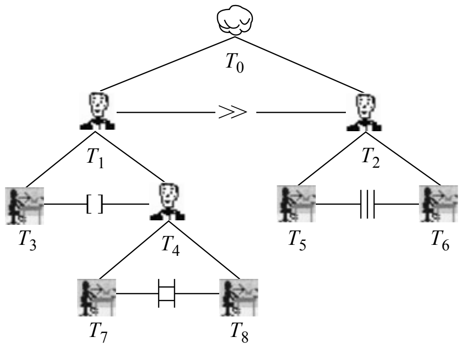

Fig. 1 Example of CTTE task model

3 Plastic user interfaces

The concept of interface plasticity is defined by Thevenin et al. [1] as the capacity of a user interface to adapt itself or to be adapted to the context of use (environment, platform and user profile) while preserving usability.

3.1 Some characteristics of plasticity

The capability of a user interface to adapt itself or to be adapted is possible only if the user interface is adaptable and adaptive. These two relevant characteristics, i.e., *adaptivity* and *adaptability*, relate to plasticity. 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. Following [30], *Adaptability* is the capacity to change and adaptivity is the capacity to accept changes. According to [31], a user interface is adaptable when it allows a user or a designer the possibility to adapt it. It is adaptive when it adapts itself to occurring changes.

A user interface is often designed for predefined interaction devices and platforms, and each platform and environment is characterized by its own interaction modes. Plasticity is an important property to ensure the safety and usability of interactive systems which is one of ISO/IEC 9126-1 usability quality of service criteria [2]. It aims at supporting UI adaptation to several running situations by re-modeling a part of or the whole user interface. This is particularly useful to continue interacting with the system even if failing situations occur [3]. The continuity of the interaction is an important concern to be taken into account when a user interface is designed.

3.2 Previous work

Nowadays, it is well accepted that plasticity is an important characteristic [1], to be addressed during the user interface design process. Studying plasticity of user interfaces has drawn the attention of researchers. The work [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.

Previous research work addressing plasticity suggested technical solutions to address the problem of user interface adaptation.

One can mention solutions based on specific software developments at the adaptation level like the Adaptive toolkit ACE [32], FRUITS [33], Multimodal Widgets [34]. The adaptation capabilities are implemented at software level.

Beside software, some techniques promoting the implementation of interaction adaptation at runtime as a feature of the operating system, like in *ToolGlasses* [35] or FACADE [36] were suggested. The objective of these approaches is to support adaptation of the UI at runtime without any modification of its internal code. The key idea consists in embedding, inside software components, integrated to the software part of the user interface, the various characteristics of interaction modalities and devices present on different platforms. The operating system integrates these adaptive components. Here, adaptive user interfaces (the user is in charge of the adaptation by supplying these components) are promoted but not adaptable interfaces.

In 2003, the European project *Cameleon* [4] initiated by the Human Computer Interaction community proposed a consensual reference framework defining the development process of plastic user interfaces in order to cover all aspects of human computer interaction (both software and human aspects). This reference framework promotes handling of plasticity characteristics, at early stages of the user interface development process. The key idea consists in designing a user interface once and equipping it with several interaction modes and devices. Successive suitable transformations to fit the characteristics (interaction techniques, modalities, environment, etc.) are applied on the designed user interface. These transformations lead to various target platforms. Several techniques and toolboxes result from this approach. The most significant ones are USIXML [7], COMET [37], WAHID [38] and MultiModel Widgets [39]. USIXML is an XML-based framework, where a user interface is specified once and multiple implementations, for various target platforms, may be produced from this description. In the COMET environment [37], 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 [38] and MultiModel Widgets [39].

The adaptation of presentation and rendering was also studied. The AMF [40] approach defines a set of interaction patterns in a multi-agent setting. Each pattern is defined so that it adapts user interface input/output according to the described interaction mode. In a similar approach, the user interface Module Adaptation approach of [41] supports GUI adaptation to the context of use (platform, environment, user profile). The user rearranges the application's user interface. Hiding and/or showing presentation elements of the interface are performed according to a matching algorithm with user interaction preferences and requirements.

More recently, taking into account the *context of use* has been addressed, particularly the user profile and environment. In [42], a UI is adapted according to user profiles, stored in an ontology. User profiles are also in the center of nomadic adaptable UI design [43, 44], exploited in MAGALLEN [45] to synthesize UI prototypes. In the ubiquitous widgets approach [46], user interactions are captured and transmitted by specific components, called IBC (Interactor Business Component) used to implement adaptive UI widgets.

Most of the previously discussed approaches to handle plasticity focused on the software part of the user interface. They led to the definition of techniques for the adaptation of software components of the interface. Few approaches addressed interaction handling adaptation of the dialogue between the user and the interface. This last aspect is handled by the third generation approaches. In these approaches, user profiles and the environment of the user interface are taken into account for adaptation. Ontologies are used to store user profiles in approaches like [42, 45].

4 Our approach

Our approach advocates the formalization of both user task models and interaction substitutions in order to handle formal verification of plasticity of user interfaces.

4.1 Plasticity and user task models

From the overview of the different approaches to handle plasticity in user interfaces of Section 3, it appears that

- a lack of interest is paid to the design and validation of a dialogue between a user and an interface when achieving a given user task with several interaction devices and different platforms representing "the context of use" as defined by Coutaz [2];
- no existing approach has addressed the problem of formally modeling the plasticity property in order to allow user interface designers to check this property at design time.

In this paper, we consider that,

- if the adaptation of the interaction is achieved through the adaptation of the task model, usually used to describe the interaction, then plasticity would be addressed at the interaction level rather than at the software part of a user interface;
- if the task models and substitutions of interactions occurring in task models are formalized, then it becomes possible to formally verify the plasticity of a given user interface.

4.2 Plasticity seen as an explicit knowledge domain

Several approaches to address the diversity and the heterogeneity of interaction devices and modes have been proposed. These approaches rely on the definition of a set of potential interaction devices and modes that can be used as substitutes for other ones in a given situation or context. Most of the significant approaches are based on

1) modeling interaction styles using pattern descriptions similarly to the agent multi-facets approach (AMF) [6]. AMF proposes to specify different interaction techniques in a set of patterns to be used in adaptation strategies;
2) the definition and the use of a catalogue of human computer interaction development technologies following [47]. COMET [37] suggests to build a catalogue of user interface development technologies consulted by COMET at runtime to define which adaptation style applies;
3) producing target user interfaces by transformation of a source one. For example, genetic algorithms based approaches were applied in MAGALLEN [45] to produce user interfaces prototypes by a mutation mecha-

nism tuned by a given user profile. These approaches choose the suited interaction technique, mode or device among a set of different candidate ones.

Therefore, whatever is the chosen adaptation strategy (at design time or at runtime) or the adaptation mode (by a user [6, 20] or automatically [37, 45]), it is necessary to define and model the description of the different concepts needed to achieve this adaptation (user interface development models, interaction modes, interaction devices, mappings and correspondences between different interaction devices and/or modes, etc.).

We claim that domain ontologies [48, 49] are good candidates for modeling such concepts and mappings between these concepts. According to [48], domain ontologies are knowledge models that provide with an explicit specification of the concepts of a domain. They can be viewed as a dictionary of concepts and of properties that hold among these concepts [49]. The interest of ontololgies is to provide explicit semantic definitions of concepts independently of any context of use.

A first attempt to use ontological approach in plastic user interface design was carried out by [7]. It consists of integrating ontology reasoning in USIXML to be able to describe multi-path development approaches. The ontology provides definitions of concepts manipulated by different models of a user interface design according to the *Cameleon* framework [4]. This approach exploits terminological aspects available in the ontology. A more recent approach uses domain ontologies to adapt information system user interfaces to a user profile in the transportation domain [42].

4.3 Two key requirements

Two characteristics of a user interface must be taken into account to ensure the plasticity of user interfaces. One relates to the implementation of the presentation software components and the other to the interaction techniques offered by the devices. These characteristics are handled in the bottom part of a user tasks model associated to a given user interface.

- *Req*$_1$– Adaptation of the implementation techniques

  At presentation level, adaptation requires that the implementation of the presentation components is supported by the used implementation technologies available in the target environment and/or platform to best adapt the user interface [2].

  Indeed, implementation techniques differ from one platform to another according to the underlying operating system, graphic display technology, interaction modes, etc. Two adaptation techniques are identified. The presentation side of the UI can be adapted either

  – by remodeling or redesigning the interface, for example, in the case of a substitution of a set of radio buttons by a menu [50];

  – or by the transfer (move) to a target platform of a part of the interface (redistribution), for example in the painter application of [51] where the color palette is implemented in a personal digital assistant (PDA) while a drawing board is implemented on a personal computer (PC) platform.

  The first adaptation technique requires to replace presentation widgets according to alternatives allowed by the underlying operating system of the target platform.

  The second adaptation technique requires

  1) to chose which part of the interface can be distributed according to interaction techniques allowed by the target operating systems (*is it possible to run this part of the UI ?*),
  2) the environment of execution (*does the environment of the target platform allow the execution? are light, sound, etc. available ?*) and
  3) the opportunity to migrate a part of the whole application (*for example in the painter application, the color palette is the part of the UI which can be separated from the drawing board as it is the case in the real world*).

- *Req*$_2$– Adaptation of the interactions offered by the devices

  At interaction level, adaptation requires the knowledge of how a given interaction device can be replaced by another one in order to allow users to pursue interacting with the system. This interaction shall continue even if failures occur [3].

  The physical actions offered by an interaction device may differ from one interaction device to another, but they may often *produce the same effect on the user interface components*. In fact, the effect produced by a physical action corresponds to an abstract interaction that may be realized by different interactions offered by the physical device. For instance, a click on the left mouse button produces the same effect as a press on the keyboard ENTER key. From the abstract task point of view, it corresponds to the *GO* action as defined in a canonical abstract interaction [52].

  The replacement of an interaction of a source device by the ones of another target device requires the substitution of the physical actions performed within the source device by those of the target device which produce the same effects on the user interface. As a consequence, to handle plasticity, matching mechanisms between an action and the effect it produces on the user interface are needed.

4.4 Our approach

As stated in Section 2, most of the work in the field of UI plasticity focussed on the first requirement (Req_1) related to the implementation techniques adaptation which gained a relative maturity.

We focus on the second requirement (Req_2) related to the adaptation of the interaction where we consider that a lack of interest still exists.

Moreover, the design life cycle of a multi-platform interface entails building a task model for each platform. The abstract part (*top level*) of this task model remains identical for each platform while the concrete part (*bottom 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 platform and/or 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].

In the context of user interface plasticity, our approach aims to address the problem of adaptation of the interaction to different platforms. When several strategies of interaction techniques are often allowed to implement a given task (*top level of a task model*) on several platforms, we propose to use formal techniques to check if these strategies are equivalent. We compare task models corresponding to each strategy, leading to compare user behavior in each strategy.

To achieve this goal, we represent each task model by an automaton, i.e., a labelled state transition system, and check equivalence (more precisely, *behavioral equivalence*) of task models by bi-simulation relationship. Since the interaction devices used in these different adaptation strategies are often different, the obtained labelled transition-systems have different sets of labels (interaction events). This leads to compare different behaviors expressed by labelled state transition systems with different sets of labels. However, the classical definition of the bi-simulation relationship of Milner in [53] does not handle different sets of labels. Thus, it cannot be used directly to compare different labelled state transition systems issued from two different task models associated to different platforms. Therefore, to compare different user task models associated to different user interfaces platforms, the labelled state transition systems need to be reworked before the classical bi-simulation relationship is applied.

In our approach [54], we advocate the definition of an ontology model including descriptions of user tasks, interaction modes and interaction devices. The subsumption and equivalence relationships are used to define substitutable tasks, interaction modes and devices.

This approach [54] relies on equivalence checking. It consists of two main steps:

- First, unifying task models using a domain knowledge model expressing semantic equivalences between interaction devices and/or interaction modes;
- Second, equivalence checking of unified task models by checking observational equivalence by means of weak bi-simulation relationship.

By reworking task models, we mean the identification of equivalent interactions (those offered by the physical devices and/or their composition). Because of the continuous evolution and the emergence of new interaction devices and the associated interaction techniques, we *advocate the use of domain ontologies to formalize equivalence matchings between device interactions and/or their compositions*. These matchings are expressed by explicit links between physical actions allowed by an interaction device and all the possible effects they may produce. Moreover, we also require to categorize these effects at the abstract level of interaction in order to characterize each physical interaction by an abstract one. The availability of this tacit knowledge for software adapters may enhance automatic UI adaptation and even self-adaptation at runtime.

5 Comparing labelled transition systems

The proposed approach relies on the capability to compare labelled transition systems in order to establish behavioral equivalence of such systems. Different relationships have been introduced in the literature to define various kinds of *lts* comparison from a behavioral point of view. Indeed, simulation is used to link a *lts* that includes the behavior of another *lts*. Symmetrically, bi-simulation defines a binary equivalence relation on *lts* states. This equivalence may be an exact equivalence through strong bi-simulation and observational or behavioral with weak bi-simulation. In general, weak relationships are used to identify labelled transition systems that share a same behavior.

In this section, we recall the basic definitions related to labelled transition systems and their comparison. These definitions were set up by the fundamental and seminal work of Milner [28, 53] for observational equivalence.

5.1 Basic definitions

Definition 1 A labeled transition system L is a structure $L = \langle S, s_0,\ E,\ \longrightarrow\rangle$ where S is a set of states, $s_0 \in S$ denotes an initial state, E is a set of labels and $\longrightarrow\ \subseteq S \times E \times S$ is a transition relation between states.

Notations When specifying systems by labelled transition systems, labels denote actions and the specific label $\tau \in E$ is used to denote internal actions, i.e., non observable actions. We note E^* to represent the set of all possible sequences of labels of E and LTS as the set of all labelled transition systems.

Definition 2 A transition (s, e, s'), also written, $s \xrightarrow{e} s'$ denotes the transition from state s to state s' with label e. E^* is the set of all sequences of labels. Let $t = e_1, e_2, e_3, \ldots, e_n \in E^*$ be a sequence of labels. A path or a trace $s_1 \xrightarrow{t} s_n$ is a sequence of transitions of the form $s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \ldots s_{n-1} \xrightarrow{e_{n-1}} s_n$. It can be a finite trace or an infinite one. We write $\xrightarrow{e^\star}$ to denote a sequence of zero or more transitions of label e in E.

We also write $s \overset{e}{\Longrightarrow} t$ the trace $s \xrightarrow{\tau^*} s' \xrightarrow{e} t' \xrightarrow{\tau^*} t$.

5.2 Relations on states

Let $T = \langle\ S, s_0,\ E,\ \longrightarrow\rangle$ be a *lts*.

Definition 3 (Simulation relationship on states) Let $\prec\ \subseteq S \times S$ be a binary relationship on states. $\prec$ is a *Simulation* when for any states $p, p', q \in S$ and for any $a \in E$, if $(p, q) \in \prec$ and $p \xrightarrow{a} p'$, then there exists $q' \in S$ such that $q \xrightarrow{a} q'$ and $(p', q') \in \prec$.

We say that state q simulates p according to $\prec$. Note that simulation is not a symmetric relationship.

Definition 4 (Weak simulation relationship on states) Let $\sim\ \subseteq\ S \times S$ be a binary relationship on states. $\sim$ is a *weak simulation* when for any states $p, p', q \in S$ and for any $a \in E$, if $(p, q) \in \sim$ and $p \overset{a}{\Longrightarrow} p'$, then there exists $q' \in S$ such that $q \overset{a}{\Longrightarrow} q'$ and $(p', q') \in \sim$.

We say that state q simulates p according to $\sim$. Note that weak simulation is not a symmetric relationship.

Definition 5 (Strong bi-simulation relationship on states) A bi-simulation is a symmetric relation of simulation. Let $\preceq\ \subseteq\ S \times S$ be a binary relationship on states and $p, q \in S$ such that $(p, q) \in \preceq$.

$\preceq$ is a strong bi-simulation relationship if

- $p \xrightarrow{a} p'$ implies that there exists $q' \in S$ such that $q \xrightarrow{a} q'$ and $(p', q') \in \preceq$;
- $q \xrightarrow{a} q'$ implies that there exists $p' \in S$ such that $p \xrightarrow{a} p'$ and $(p', q') \in \preceq$.

Definition 6 (Weak bi-simulation relationship on states) Let $\approx\ \subseteq\ S \times S$ be a binary relationship on states and $p, q \in S$ such that $(p, q) \in \approx$.

$\approx$ is a weak bi-simulation relationship if

- $p \xrightarrow{a} p'$ implies that there exists $q' \in S$ such that $q \overset{a}{\Longrightarrow} q'$ and $(p', q') \in \approx$;
- $q \xrightarrow{a} q'$ implies that there exists $p' \in S$ such that $p \overset{a}{\Longrightarrow} p'$ and $(p', q') \in \approx$.

5.3 Extensions to labelled transition systems

The previous definitions are extended to labelled transition systems. Let $lts = \langle S, s_0\ E,\ \longrightarrow\rangle$ and $lts' = \langle S', s'_0\ E,\ \longrightarrow\rangle$ be two labelled transition systems with the same set of labels E and $S \cap S' = \emptyset$.

Informally, this extension of these relations to labelled transition systems consists in requiring the existence of a corresponding relation on the initial states of these labelled transition systems.

Definition 7 (Simulation relationship on *lts*) Let $\prec\ \subseteq S \times S'$ be a simulation relationship. We define $\prec_{lts} \subseteq LTS \times LTS$ as a simulation relationship between two labelled transition systems.

Then $(lts, lts') \in \prec_{lts}$ if $(s_0, s'_0) \in \prec$. Informally, the relation $\prec_{lts}$ on *lts* is a *simulation* relationship if it is possible to build a state simulation relation ($\prec$) that includes their initial states.

We say that lts' simulates lts.

Definition 8 (Weak simulation relationship on *lts*) Let $\sim\ \subseteq\ S \times S'$ be a weak simulation relationship. We define $\sim_{lts} \subseteq LTS \times LTS$ as a weak simulation relationship between two labelled transition systems.

Then $(lts, lts') \in \sim_{lts}$ if $(s_0, s'_0) \in \sim$. Informally, the relation

$\sim_{lts}$ on *lts* is a *weak simulation* relationship if it is possible to build a state weak simulation relation ($\sim$) that includes their initial states.

We say that lts' weakly simulates lts.

Definition 9 (Strong bi-simulation relationship on *lts*) Let $\preceq \subseteq S \times S'$ be a strong bi-simulation relationship.

We define $\preceq_{lts} \subseteq LTS \times LTS$ as a strong bi-simulation relationship between two labelled transition systems.

Then $(lts, lts') \in \preceq_{lts}$ if $(s_0, s'_0) \in \preceq$.

Informally, the relation $\preceq_{lts}$ on *lts* is a *strong bi-simulation* relationship if it is possible to build a state strong bi-simulation relation ($\preceq$) that includes their initial states.

We say that lts' and lts are bi-similar.

Definition 10 (Weak bi-simulation relationship on *lts*) Let $\approx \subseteq S \times S'$ be a weak bi-simulation relationship.

We define $\approx_{lts} \subseteq LTS \times LTS$ as a weak bi-simulation relationship between two labelled transition systems.

Then $(lts, lts') \in \approx_{lts}$ if $(s_0, s'_0) \in \approx$. Informally, the relation $\approx_{lts}$ on *lts* is a *weak bi-simulation* relationship if it is possible to build a state weak bi-simulation relation ($\approx$) that includes their initial states.

We say that lts' and lts are weakly bi-similar.

6 Revisiting *lts* comparison

The previously defined relationships support the comparison of labelled transition systems that act on the same set of labels. The situation where the need of comparing labelled transition systems with different sets of labels may occur. For example, interactive systems, addressed in this paper, are one of the cases where different interaction possibilities are offered to interact with a given system.

In this section, we define another bi-simulation relationship that relaxes the classical definition. It relates labelled transition systems with different sets of labels.

The proposed definition relies on the introduction of a relation on labels. This relation links pairs of labels. It is exploited to transform the labelled transitions systems to be compared labelled transition systems by substituting labels so as they get the same set of labels.

6.1 Rewriting labels and transforming labelled transition systems

Let $lts = \langle S, s_0\ E, \longrightarrow\rangle$ and $lts' = \langle S', s'_0\ E, \longrightarrow\rangle$ be two transition systems such that $S \cap S' = \emptyset$, $E \not\subseteq E'$ and $E' \not\subseteq E$.

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

Definition 11 (Bi-directional relation on labels) $\Gamma \subseteq (E - E') \times (E' - E)$ is a relation on labels of two labelled transition systems. It satisfies

$\forall\, \alpha \in E - E', \exists\, \beta \in E' - E$ we have $(\alpha, \beta) \in \Gamma$;
$\forall\, \beta \in E' - E, \exists\, \alpha \in E - E'$ we have $(\alpha, \beta) \in \Gamma$.

Left and right projection functions $Proj_l$ and $Proj_r$ are associated to Γ. Informally, the relation Γ on labels defines a total relation on the labels that do not belong to $E \cap E'$, i.e., the labels that are not shared by the two labelled transition systems.

Definition 12 (Rewriting function on labels) The function $\Phi : E \times E' \longrightarrow (A \cup E \cup E' \cup \{\tau\})$ on labels of two labelled transition systems is defined by

$\forall\, (\alpha, \beta) \in \Gamma\ \ \exists\, \gamma \in A \cup E \cup E' \cup \{\tau\}$ such that $\Phi(\alpha, \beta) = \gamma$.

Four main rules can be associated to the definition of the rewriting function.

1) Substitution $\exists\, e \in A$ such that $\Phi(a, b) = e$ to denote that labels a and b are replaced by a new label e in A.
2) Right replacement : for $a \in E\ \exists\, b \in E'$ such that $\Phi(a, b) = a$ to denote that a label $b \in E'$ of lts' is replaced by a label $a \in E$ of lts.
3) Left replacement : for $b \in E'\ \exists\, a \in E$ such that $\Phi(a, b) = b$ to denote that a label $a \in E$ of lts is replaced by a label $b \in E'$ of lts'.
4) Hiding : for $a \in E,\ b \in E'$ with $\Phi(a, b) = \tau$, we denote the case of a pair of labels that should be hidden on both labelled transition systems lts and lts' after rewriting.

Definition 13 (Transforming labelled transition systems) The 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 with

- same sets of states $S^{\top} = S$ and $S^{\top'} = S'$;
- same initial states $s_0^{\top} = s_0$ and $s_0^{\top'} = s'_0$;
- sets of labels where different labels are rewritten thanks to the Φ rewriting function $E^{\top} = (E - Proj_l(\Gamma)) \cup A \cup \{\tau\}$ and $E^{\top'} = (E' - Proj_r(\Gamma)) \cup A \cup \{\tau\}$;
- transition relations are redefined with 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

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

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

6.2 Comparison of labelled transition systems with different sets of labels

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 11,
- Φ is a label rewriting function according to Definition 12.

Definition 14 (Relational simulation relationship on *lts*) $\prec^{\Gamma,\Phi}_{lts} \subseteq LTS \times LTS$ is a relational simulation relationship on labelled transition systems if there exists a simulation relationship on labelled transition systems between the transformed *lts*. We write

$$(lts, lts') \in \prec^{\Gamma,\Phi}_{lts} \Longleftrightarrow (lts^{\top}, lts^{\top'}) \in \prec_{lts}.$$

Definition 15 (Relational weak simulation relationship on *lts*) $\sim^{\Gamma,\Phi}_{lts} \subseteq LTS \times LTS$ is a relational weak simulation relationship on labelled transition systems if there exists a simulation relationship on labelled transition systems between the transformed *lts*. We write

$$(lts, lts') \in \sim^{\Gamma,\Phi}_{lts} \Longleftrightarrow (lts^{\top}, lts^{\top'}) \in \sim_{lts}.$$

Definition 16 (Relational strong bi-simulation relationship on *lts*) $\preceq^{\Gamma,\Phi}_{lts} \subseteq LTS \times LTS$ is a relational strong bi-simulation relationship on labelled transition systems if there exists a strong bi-simulation relationship on labelled transition systems between the transformed *lts*. We write

$$(lts, lts') \in \preceq^{\Gamma,\Phi}_{lts} \Longleftrightarrow (lts^{\top}, lts^{\top'}) \in \preceq_{lts}.$$

Definition 17 (Relational weak bi-simulation relationship on *lts*) $\approx^{\Gamma,\Phi}_{lts} \subseteq LTS \times LTS$ is a 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

$$(lts, lts') \in \approx^{\Gamma,\Phi}_{lts} \Longleftrightarrow (lts^{\top}, lts^{\top'}) \in \approx_{lts}.$$

From the definitions of the relationships introduced in the previous definitions, it becomes possible to compare labelled transition systems with different sets of labels.

6.3 About Γ and Φ

The definition of Γ and Φ are of great importance to define label mappings and label rewritings. The formal setting described above requires the existence of a

1) relation Γ between the labels associated to one labelled transition system which do not occur in the other one;
2) transformation function Φ which associates to each pair of labels in Γ another label different from those of the two considered labelled transition systems.

The definitions of Γ and Φ given above are minimal definitions. Strengthening these definitions with additional constraints and properties remains possible. This strengthening shall preserve the capability to rewrite the labels. In this case, new properties on the labels of labelled transition systems and thus of the labelled transition systems themselves can be deduced.

7 A formal model for designing plastic interfaces

The classical simulation and bi-simulation relationships are defined on a single set of labels. Their definitions compare transitions with the same labels. The need of comparing systems with close or equivalent behaviors that do not use the same transition labels may occur in several situations particularly in the case of system substitution and thus in the case of plastic user interface.

Systems for which relational simulation or relational bi-simulation relationships are useful are those systems whose behavior may lead to equivalent states but which use different actions. Plastic interactive systems described in Section 3 correspond to such systems. These systems use different interaction modes and devices to achieve the same user tasks. As stated in Section 2, by plasticity we mean the capability to achieve a given interactive task using different interactive modes and/or devices. In other words, two interactive systems may realise the same action using different interaction modes or devices. Relational bi-similarity can be used to check that these two systems are *equivalent* modulo the relations on the labels.

In the remainder of this section, we put into practice the relational bi-simulation and show how it is set up to check the plasticity property of interactive systems. An ontology is defined in order to semantically model relations on labels. We also give a stepwise methodology to support this checking process.

7.1 Task models as labelled transition systems

A task model allows a designer to describe tasks to be supported by the designed user interface. A task model gives the details of both a static aspect which corresponds to the structure of the task (a decomposition tree in our case) and a dynamic aspect which corresponds to behavior of the user and the system (labelled transition systems in our case) during the achievement of the task.

Various experiences reported in the literature have represented task models by labelled transition systems. For instance, in [13] *lts* have been derived from a task model to simulate user behavior in order to verify task achievement. In [55], *lts* built from a task model are used to formally verify properties of multi-modal user interfaces. In both experiences, the set of states is built from attributes available in the user interface components and the set of transitions is built from user actions.

A task model expresses a hierarchical decomposition of the task into subtasks up to physical actions (keystroke). At the same time, it specifies a temporal interleaving of these subtasks and actions (consecutive, parallel, alternative, iterative composition operations). In other terms, it describes the behavior of the user interacting with the interface entailing a modification in its state. This behavior is captured by a formal model described as a labelled state-transition system made of user actions composing the task and of the interface reaction. User actions are modeled by transitions and the interface reactions are denoted by the states of the system.

Task models written in CTT task modeling notation of Section 3 are particularly suitable to be represented by a labelled transition system. A *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 a *lts* derived from the task model example depicted in Fig. 1 of Section 2. Its set of labels $\{T_3, T_4, T_5, T_6, T_7, T_8\}$ is built from the user actions of the task model (the user actions are in the leaves of the task tree of the Fig. 1) and the set of states $\{S_0, S_1, \ldots, S_8\}$ is composed of some relevant user interface components attributes impacted by these actions.

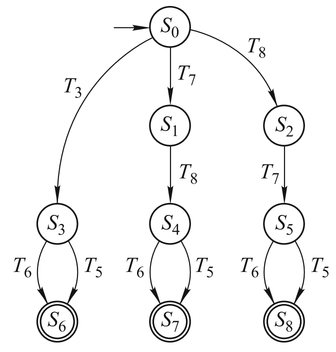

Fig. 2 The *lts* representing task model

7.2 An ontology of interaction

The labels of a transition system derived from an interactive task model (e.g., a CTT task model) represent actions performed on the interactive system either by the human (user) or by the machine. The semantics of these actions can be defined within and ontology which also gives a label classification.

We define an ontology providing hierarchical categories of interactive actions or devices. As shown on the UML class diagram model depicted on Fig. 3, the basic concepts of our ontology are the *interaction device*, the *interactive task* and the *user interaction* respectively denoted by DEV, INTTASK, INTR in Fig. 3.

7.2.1 Basic concepts

1. The interaction device (DEV) concept defines a hierarchical category of devices, well known in the human computer interaction domain, which may be used to perform tasks. This category is inspired from the taxonomy of interaction devices defined by Buxton [56], Card et al. [57] and Frohlich [58].
2. The interactive task (INTTASK) concept refers to the user interaction at abstract level in the same manner as "abstract interaction" defined in [52] (select, copy, text input, etc.).
3. The user interaction (INTR) concept describes patterns of interaction techniques as well as those defined in [3] for user interaction reconfiguration. They are similar to those used to define interaction strategies for AMF agents [6] and to those defined for input adaptability in the ICON toolkit [59]. A user interaction can be basic

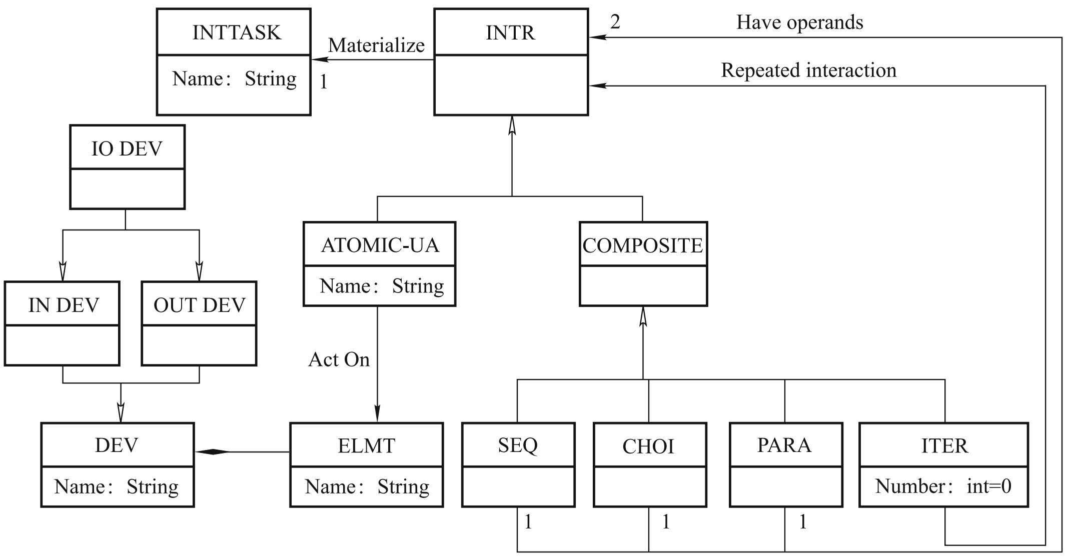

Fig. 3 The ontology model represented as a UML class diagram

or composite.

- A basic interaction (ATOMIC-UA) defines an atomic physical *user action* (keystroke). It is linked to an *element* (ELMT) of the interaction device concept on which it has effects. A user action may affect the behavior of the user interface. For example, a click (atomic user action) on a left button (element) of the mouse (interaction device) triggers the GO behavior of the user interface.
- A composite interaction (COMPOSITE) describes the case of an interaction composed of a set of user actions using the composition operators of sequence (SEQ), concurrence (PARA), choice (CHOI) and iteration (ITER). Each interaction materializes a way to perform an interactive task with a set of user actions offered by the interaction devices available on a user interface platform.

7.2.2 Basic relations

Our ontology acts like a dictionary of interaction techniques that formalize a set of patterns defining different implementations of a single interactive task, depending on various interaction devices existing in the human computer interaction domain. It offers different kinds of relations between *user actions*. As usual for ontologies, two main relationships between concepts are introduced: *equivalence* and *subsumption*.

1) Equivalence This relation means that two user actions linked to an element of an interactive device (or two sets of user actions based on two user interactions) have the *same effects* on an interactive system. In other words, the equivalence relationship expresses that an interactive task can be performed by one of the two single *user actions* or one of two user interactions.

2) Subsumption This relationship defines a hierarchical relationship encoded by inheritance. When a user action subsumes a set of other user actions *Us*, then the effect of this user action entails the ones belonging to this set of actions *Us*. Subsumption relation expresses the fact that an interactive task can be performed either by a single user action or by a set of user actions composing a user interaction.

7.3 Rewriting rules for labels

When rewriting labels of *lts* derived from *user actions*, the relation linking an interactive task and the user interaction is exploited as follows.

- $Rule_1$ If two single user actions are equivalent, they may be rewritten with the label corresponding to the interactive task they perform.
  Example 1 *Press Enter key* and *click left button* are equivalent since both of them correspond to a GO behavior on the UI. They may be rewritten as the GO label corresponding to the interactive task they perform.
- $Rule_2$ If a single user action u subsumes a set of user actions *Us* composing a user interaction, then this single user action may be rewritten with as the label corre-

sponding to the interactive task it performs. Moreover, all the user actions of the set of user interaction *Us* is rewritten with a single label corresponding to the same label as the one of interactive task.

Example 2   The user action *point tablet screen* entails the GO behavior equivalent to the ones entailed by the sequence of user actions *move mouse SEQ click*. We say that the former subsumes the later and both single action *point tablet screen* and the interaction may be rewritten as the single label GO.

- $Rule_3$ If a pair of two sets of user actions (two user interactions) are equivalent they may be rewritten in the same manner as a single label corresponding to the interactive task they perform.

  Example 3   The composite user interaction *press mouse button down* in parallel with *move mouse* followed by *release mouse button up* defines the selection of a set of icons on a screen. The same behavior can be obtained by the composite user interaction performed by the user action *press of the shift key down* in parallel with the *press of direction key* followed by the *release of the shift key up*. This means that the two composite user interactions are equivalent. Therefore, both of them may be rewritten to the single label *Multi-Selection* corresponding to the interactive task they perform.

7.4 Methodology

The whole material required to check the plasticity of a user interface is now set up. Checking the plasticity property consists in checking that two different interactive systems allows a user to achieve the same tasks using different interactive modes and/or different devices.

• Basic principle   The proposed approach consists in formalizing the considered interactive systems by labelled transition systems, and then checking a relational bi-simulation on these two systems, provided that a relation on labels of their corresponding labelled transition systems is available.

To check the plasticity property of a pair ($Syst_{source}$, $Syst_{target}$), we have set a stepwise methodology consisting in the following steps:

1) Design   Design the pair ($lts_{source}, lts_{target}$) of labelled transition systems formalising ($Syst_{source}, Syst_{target}$).
2) Irrelevant action identification   For ($lts_{source}, lts_{target}$), identify the possible 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   Using a relation on labels of the *lts*, the labels of lts_{source} and lts_{target} that are different are rewritten. At this stage, the two lts_{source} and lts_{target} have the same labels.
4) Checking   Check weak bi-simulation between the obtained *lts*.

In other words, two interactive systems satisfy the plasticity properties if they are linked by a relational bi-simulation relationship according to a given relation on labels.

8 Validation on the case study

This section shows how our approach to check user interface plasticity applies on two case studies. The first one addresses the case of a web interface to send an SMS (short message sending) and the second one deals with a mobile casual game application. In both cases, the four-step methodology we defined in the previous section (design, irrelevant action identification, rewriting, checking) is deployed.

8.1 Desktop web application UI adapted to smart phone

Our case study concerns an interface of a web application for sending an SMS (short message sending). The user interface is composed of a set of six (06) web forms to be filled in order to send an SMS. First, the user opens a session to login to his own space (first web form), then he composes his message (second and third web forms) and sends the composed SMS (forth and fifth web forms). Finally he exits his own space (sixth web form). The CTT task model depicted in Fig. 4 describes the different user actions involved by the task consisting in sending an SMS. The following subtasks are introduced.

1) *Login* (subtask TA) where a user introduces his/her login identifier (T_1) and his/her password (T_2) in any order and submit these two entered values (T_3).
2) *Compose* is devoted to build the SMS (subtask TB). The user launches a message editor (T_4), edits his/her message (T_5) by typing the text of the message (T_8), enters the phone number of the recipient (T_7) (in any order) and then decides whether he/she sends the written message (T_9), saves it (T_{10}) or cancels it (T_{11}).
3) *Send* task is defined to model sending of the composed message. The user requests a send action to the system (T_{12}) and confirms the transaction (T_{13}) or not (T_{14}). In case the sending of the message is not confirmed, the user chooses either to save the message (T_{15}) or to can-

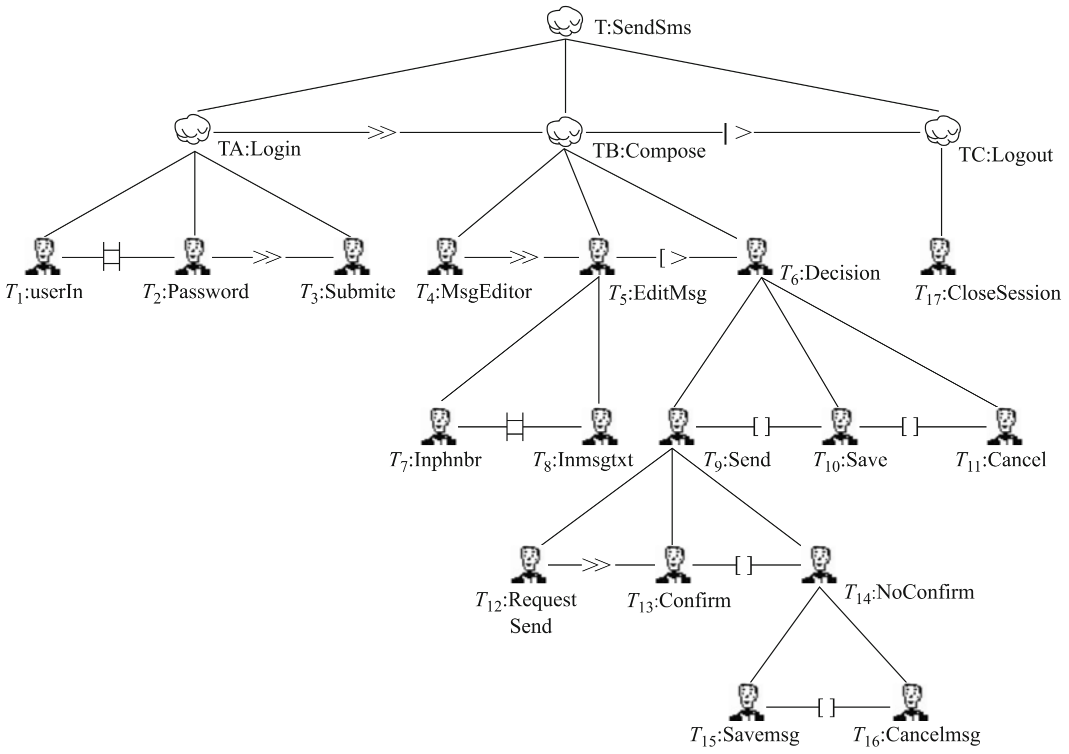

Fig. 4 Send SMS abstract task model

cel the whole operation (T_{16}).

4) Finally, *Logout* (subtask TC) is the subtask allowing the user to close the session (T_{17}).

In the rest of this section, we focus on the subtask EditMsg task corresponding to T_5 in Fig. 4. Our objective is to compare the interaction technique used to perform this task on a personal computer (a platform with a mouse and a keyboard only) on the one hand and two interaction techniques used to carry out the same task on a smart phone (a platform with a keyboard only) and on a Touch-Pad (a platform with touch screen only) on the other hand. The four-step methodology we have defined is set up.

8.1.1 Design

Let lts_{source}, $lts_{targetPh}$ and $lts_{targetTp}$ be the labelled state transition systems modeling the subtask EditMsg on the $syst_{source}$, $syst_{targetPh}$ and $syst_{targetTp}$ systems corresponding to a personal computer, a smartphone and a Touch-Pad, respectively.

The CTT task model associated to the EditMsg on the personal computer $syst_{source}$ is depicted in Fig. 5. This task model is compared to the two identified target task models.

The first task model corresponds to the interaction task performed on a smart phone $syst_{targetPh}$. The subtask EditMsg for a smartphone is described by the CTT task model of Fig. 6.

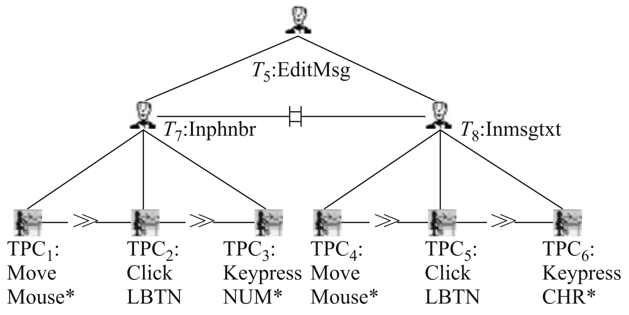

Fig. 5 The EditMsg task model of the interaction on the PC platform

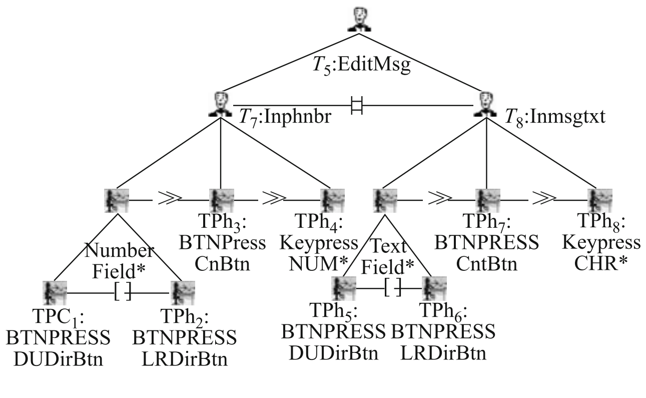

Fig. 6 The EditMsg task model of the interaction on a smart phone platform

Last, the second target task model represents the interaction performed on a Touch Pad. Figure 7 depicts the corresponding CTT task model.

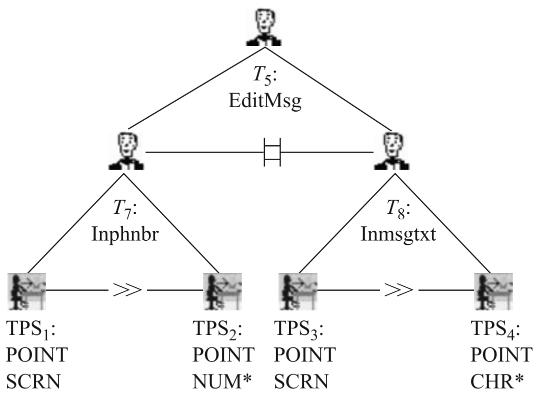

Fig. 7 The EditMsg task model of the interaction on a Touch Pad platform

In order to build the different labelled state transition systems corresponding to each platform, we associate a label for each user action available in each of the CTT task models.

Table 1 describes all the correspondences between user actions of the CTT task models and the labels in the corresponding *lts*.

Table 1 Mapping between user actions and corresponding labels

CTT's actions	Signification	*lts*'s labels
KeyPress CHR	Press the keyboard's character key	e_1
Point CHR	Point a character virtual key on a tactile screen	e_2
KeyPress NUM	Press the keyboard's digital key	e_3
Point NUM	Point a digital virtual key on a tactile screen	e_4
Click LBTN	Click the mouse's left button	e_5
BtnPress DUDirBtn	Press keypad's Down/Up direction button	e_6
Move Mouse	Move the mouse	e_7
BtnPress LRDirBtn	Press keypad's Left/Right direction button	e_8
Point Scrn	Point the tactile screen	e_9
BtnPress CentBtn	Press keypad's central button	e_{10}

The following labelled state transition systems are obtained for the EditMsg user task.

1) The *lts* lts_{source} representing the interaction on the personal computer platform is given by the labelled state-transition system of Fig. 8.

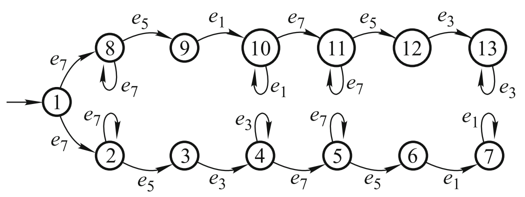

Fig. 8 The EditMsg *lts* for a PC platform

2) The task model corresponding to the interaction on a smartphone platform is represented by the labelled state transition system $lts_{targetPh}$ of Fig. 9.

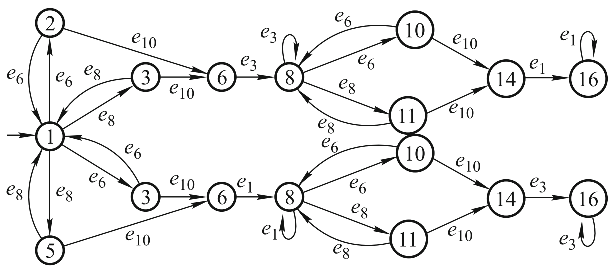

Fig. 9 The EditMsg *lts* for a smartphone platform

3) Finally, the labelled state transition system $lts_{targetTp}$ representing the interaction on the Touch Pad platform is represented in Fig. 10.

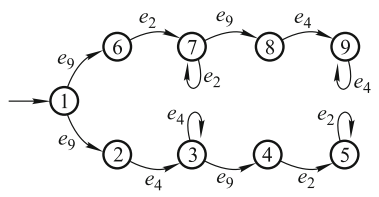

Fig. 10 The EditMsg *lts* for a Touch Pad platform

At this stage we obtain two pairs $(lts_{source}, lts_{targetPh})$ to compare the interaction used on a personal computer with the one on a smartphone and $(lts_{source}, lts_{targetTp})$ to compare personal computer interaction with the interaction on a Touch Pad.

In the remaining steps, we will address the case of $(lts_{source}, lts_{targetPh})$ only to compare the interaction used on a personal computer with the one on a smartphone.

8.1.2 Irrelevant actions identification

The next step identifies the possible internal actions considered as non relevant to perform the suited interaction. The labels corresponding to these actions in both lts_{source} or $lts_{targetPh}$ are set to τ.

The set of actions {*MoveMouse, BTNPress DUDirBtn, BTNPress LRDirBtn*} iterated on the defined labelled state transition system is not relevant for the interaction. Indeed, the presence of these iterated actions in the CTT task model expresses a cursor movement to reach a target field in a given form. Moving a cursor can be done repeatedly without any relevant effect from interaction point of view. So, only one move of the cursor is considered and the occurrences of labels of the set $\{e_7, e_6, e_8\}$ are set to τ. Another pair of labelled state transition systems (lts_{Pct}, lts_{Pht}) is obtained after irrelevant actions removal.

The labelled state transition system lts_{Pct} depicted in Fig. 11 corresponds to the lts_{source} (Fig. 8) where the iterated occurrences of the label e_7 (transitions on states 2, 5, 8 and 11) are set to τ.

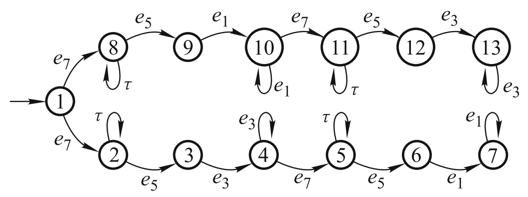

Fig. 11 Irrelevant actions set to τ for the PC platform

The labelled state transition system lts_{Pht} of Fig. 12 corresponds to the labelled state transition system $lts_{targetPh}$ (Fig. 9) where the iterated occurrences of labels e_6 and e_8 are set to τ.

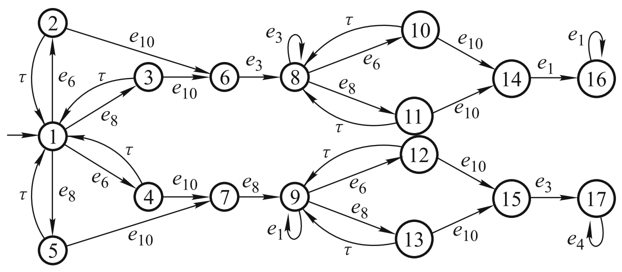

Fig. 12 Irrelevant actions set to τ for the smartphone platform

8.1.3 Rewriting

The set of labels of the labelled state transition systems lts_{Pct} and lts_{Pht} obtained after removing irrelevant actions are still different sets. A rewriting step, exploiting relations on labels borrowed from the ontology is required to obtain two labelled state transition systems with a single set of labels.

Let E and E' be the two sets of labels of lts_{Pct} and lts_{Pht} respectively.

Let $LabDiff$ be the set of different labels of lts_{Pct} and lts_{Pht}. It is defined by

$$LabDiff = (E \cup E') - (E \cap E') = \{e_5, e_6, e_7, e_8, e_9, e_{10}\}.$$

Let E_p be the set of labels of lts_{Pct} to be rewritten.

$$E_p = E \cap LabDiff = \{e_5, e_7\}.$$

Let E'_h be the set of labels of lts_{Pht} to be rewritten.

$$E'_h = E' \cap LabDiff = \{e_6, e_8, e_{10}\}.$$

The relation Γ (see Definition 11) provided by our ontology is the equivalence and/or subsumption relationships. These relations mean that user actions and/or user interactions with the same effect (equivalent or subsumed effect) can be replaced by the interactive task corresponding to this effect.

The result of this substitution operation is a set $A = \{g, m\}$ containing two labels neither in E nor in E'. These two labels must be rewritten according to the identified ontological relation in order to get a same set of labels. Figure 13 shows instances of the ontology we have used to define such relations. When applied, the rewriting function ϕ (Definition 12) produces the labels defined in Table 2.

Table 2 Label rewriting table

User actions	lts's labels	Substitution action	ϕ Application
(Click LBTN, BtnPress CentBtn)	(e_5, e_{10})	GO	$\phi(e_5, e_{10}) = g$
(Move Mouse, BtnPress DUDirBtn)	(e_7, e_6)	MoveCursor	$\phi(e_7, e_6) = m$
(Move Mouse, BtnPress LRDirBtn)	(e_7, e_8)	MoveCursor	$\phi(e_7, e_8) = m$

In Table 2,

- the label g corresponding to the interactive task "GO" is a substitute for the ones corresponding to the user actions Click LBTN and BtnPress CentBtn. Indeed, this label represents the effect of the two actions,
- the label m corresponding to the interactive task "MoveCursor" is a substitute for the labels corresponding to the user actions Move Mouse, BtnPress DUDirBtn and BtnPress LRDirBtn.

After rewriting the labels of lts_{Pct} and lts_{Pht}, the new labelled state transition systems lts_{Pctr} are obtained. They are respectively depicted in Figs. 14 and 15.

This rewriting step produces two lts with the same set of labels $\{e_1, e_3, m, g, \tau\}$.

8.1.4 Checking

The final step checks behavior equivalence on the obtained labelled state transition systems. *Observational equivalence* between the two obtained labelled state transition systems is checked. This checking supports the behavior comparison of interactive systems that do not have the same interaction modes and/or devices. To do so we use the weak bi-simulation relationship defined in Section 5. The final result showed that lts_{Pct} and lts_{Pht} are weakly bi-similar. Thus we can formally assert that the interactions described by the task models of Figs. 5 and 6 perform the same task, and thus the devices and the interactions may be substituted. The equiva-

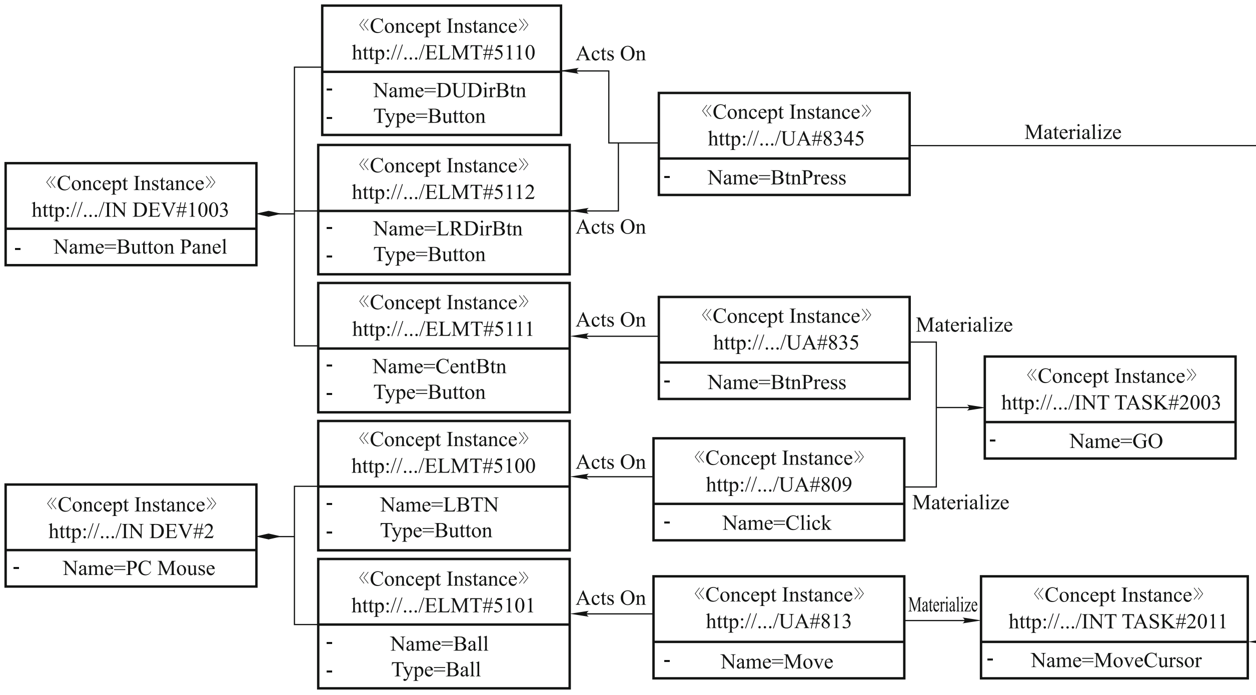

Fig. 13 Instances of our ontology concepts used to rewrite labels

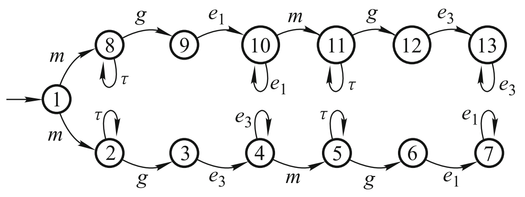

Fig. 14 The EditMsg *lts* for a PC platform after rewriting labels

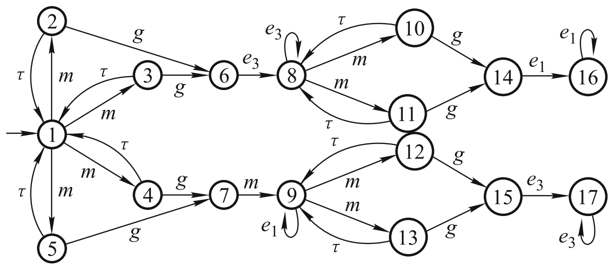

Fig. 15 The EditMsg *lts* for a smartphone platform after rewriting labels

lence of the corresponding rewritten labelled state transition systems modulo the *relational weak bi-simulation* relationship means that these two interaction techniques can substitute each other.

The previous case study has been checked within the CADP (construction and analysis of distributed processes) model checker [60]. It offers a set of tools for compiling, verifying and validating Lotos [29] process models. One of its important features is the capability to compare labelled state transition systems modulo an equivalence or preorder relation. The comparison is supported by the CADP Aldebaran bi-simulator library. If the submitted labelled state transition systems are not equivalent, a diagnostic file is generated to show the failed transition.

The labelled state transition systems of our case study are first described in LotosNt [60] (Fig. 16), a simplified version of the Lotos language, then they are transformed to full Lotos programs. Labelled sate-transitions systems in BCG (binary coded graphs) format are generated for each Lotos description. Finally the two BCG automata are compared with the bi-simulator modulo observational equivalence relation.

8.2 Smartphone game application UI adapted to PC platform

The second case study concerns an interface of a mobile casual game application called “Marble Legend”. It is a single player game where a user scores when he/she eliminates series of marbles of the same color. In this game, the user must create three or more consecutive marbles of a given color. Marbles of the same color are thrown by a shooting source (a frog). The ultimate goal of the game is to eliminate all marbles before they reach a central hole where they are sucked by a monster. The user interface of this application is composed of seven screens displayed according to the evolution of the game. First, the user selects, from the main menu, either to

```
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. 16 A section of LotosNt code corresponding to lts_{Pct}

tune the game or to play in one of the two proposed game modes: adventure or challenge modes (first screen). Once the game mode is selected (second and third screens), the user eliminates series of marbles before the time limit associated to the current game level is reached (fourth screen). When the user completes the game actions of the current level, score rates are displayed. Then, the user selects either to replay the current level (for example to increase his/her score), to move to the next level or to return back to the main menu (fifth screen). In case the user fails in this level (i.e., the marbles are sucked by the monster), he/she is forced either to retry again or to leave the current level and return to the main menu (sixth screen). At any time, for a given level, the user can pause the game. He/she can also select either to continue, restart the current level or return back to the main menu (seventh screen). The CTT task model depicted in Fig. 17 describes the different user actions involved by the task consisting in playing the marble game. The following subtasks are detailed below.

1) *Tune* (subtask TA) allows a user to parameterize the game environment: tune of sound mode (T_1), ambient music (T_2) or colors for scenes (T_3) independently.
2) *Play* is dedicated to the description of the game playing (subtask TB). The user selects one game mode (T_4) and then tries to complete the game for the corresponding level before the allowed time limit (T_5). Therefore, T_5 is decomposed as follows. The user starts the current level (launches T_9). He/she shoots colored marbles, issued from the outlet of the source (the frog), in the direction of the lines of marbles before they are sucked into the central hole representing the mouth of the monster (T_{10}). At any time, for a given level, the user may change marble color or reverse the emission order of marbles at the shooting source (T_{11}). The user can also turn the game temporary to pause (T_{12}). He/she may decide to continue, stop the game at the current level or return back to the main menu.
3) Finally, the *Interrupt* (subtask TC) is triggered when the game is interrupted either if the game level is completed, failed or stopped by the user. The user may move to the next level in case of success (T_7), retry again the current game level in case of failure (T_6) or exit the game (T_8).

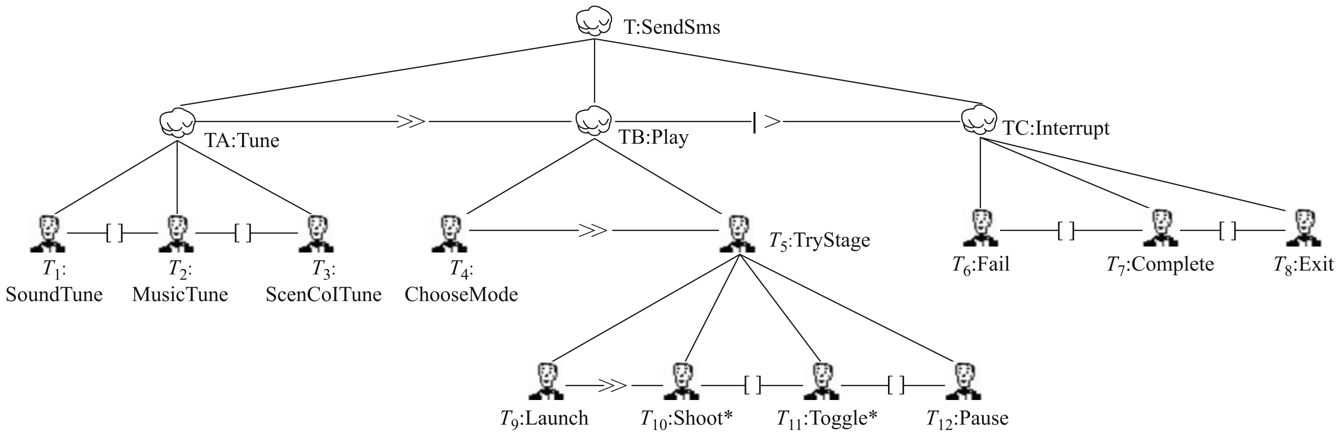

Fig. 17 "Marble Legend" game abstract task model

The remainder of this section, focuses on the TryStage subtask (T_5 in Fig. 17). Our objective is to study the plasticity of the interaction technique used to perform this task. We address the cases of a smart phone (with only a touch screen) and a personal computer (with a mouse). We show how the plasticity property of the TryStage task (T_5 in Fig. 17) can be modeled for both two platforms: the source platform is a smartphone and the target one is a personal computer. The four-step methodology we have defined is deployed for this case.

8.2.1 Design

Let lts_{source} and $lts_{targetPc}$ be two labelled state transition systems modeling the subtask TryStage (T_5 in Fig. 17).

- lts_{source} is the labelled state transition systems associated to the source system $syst_{source}$ (i.e., a smartphone platform). It models the interaction task T_5 on the smartphone platform. The CTT task model associated to the TryStage task T_5 on the smartphone $syst_{source}$ is depicted in Fig. 18.
- $lts_{targetPc}$ is the one associated to the target system $syst_{targetPc}$ (i.e., a personal computer). It models the interaction task T_5 on the PC platform. The target CTT task model for this task is depicted in Fig. 19.

In order to build the different labelled state transition systems corresponding to each CTT task model of both platforms, a label is associated to each user action of these CTT task models. Table 3 describes all these correspondences.

Table 3 Mapping between user actions and corresponding labels

CTT's actions	Signification	*lts*'s labels
Move Mouse	Move the mouse	e_1
Click LBTN	Click the mouse's left button	e_2
Click RBTN	Click the mouse's right button	e_3
Point Scrn	Point the tactile screen	e_4

From the CTT models of Figs. 18 and 19, the following labelled state transition systems are obtained for the TryStage user task T_5.

1) The *lts* lts_{source} representing the interaction on the smart phone platform is given by the labelled state-transition system of Fig. 20.
2) The *lts* $lts_{targetPc}$ representing the interaction on the personal computer platform is represented in Fig. 21.

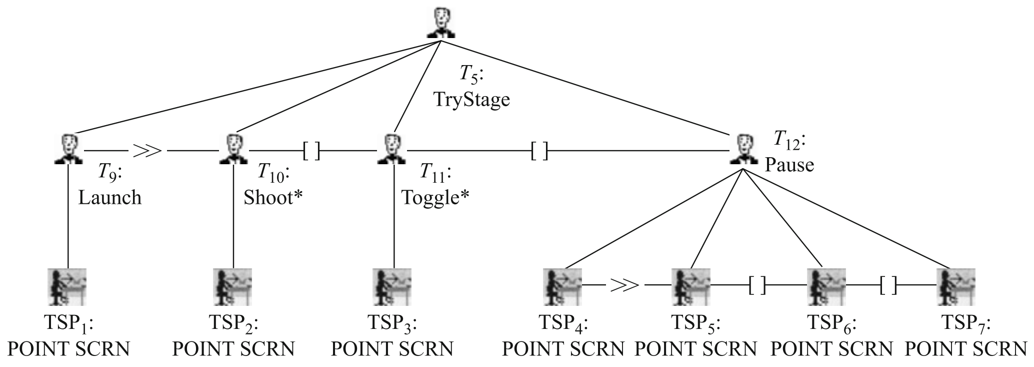

Fig. 18 The TryStage task model for a smartphone platform

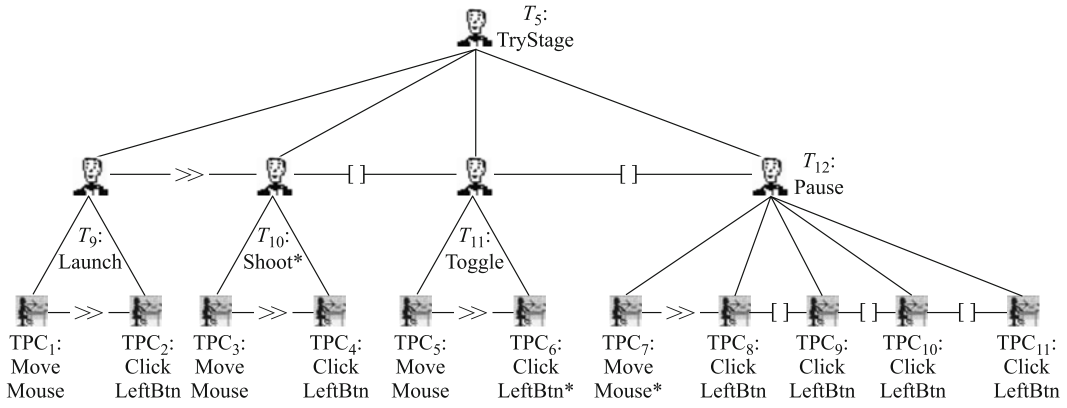

Fig. 19 The TryStage task model for a PC platform

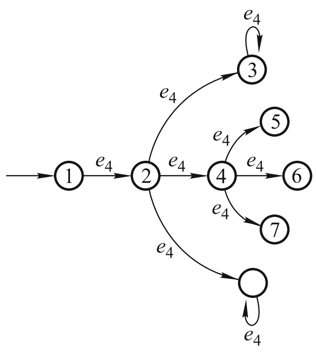

Fig. 20 The TryStage *lts* for a smartphone platform

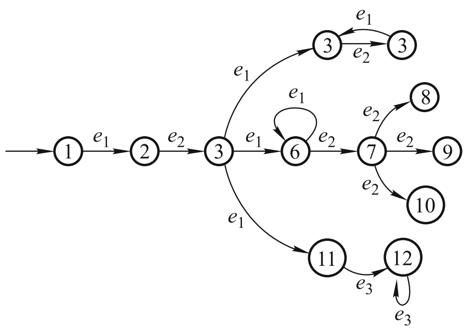

Fig. 21 The TryStage *lts* for a PC platform

At this stage we obtain a pair of $(lts_{source}, lts_{targetPc})$. We need to compare these *lts* in order to be able to check the plasticity property of the interaction task T_5 when using a smartphone or a PC platform.

8.2.2 Irrelevant actions identification

The next step identifies the possible internal actions considered as not relevant to perform the suited interaction. The labels corresponding to these actions in both lts_{source} or $lts_{targetPc}$ are set to τ. The action *Move Mouse* on the defined labelled state transition system is not relevant for the interaction. Indeed, the presence of this action in the CTT task model expresses a cursor movement to reach a target field in a given form. Moving a cursor can be done repeatedly without any relevant effect from interaction point of view. Therefore, the occurrences of the label e_1 are set to τ. Another pair of *lts* (lts_{spt}, lts_{Pct}) is obtained after irrelevant action removal. Note that the *lts* lts_{spt}, corresponding to the lts_{source} (Fig. 20) remains unchanged since it does not contain any occurrence of the label e_1. The *lts* lts_{Pct} of Fig. 22 corresponds to the *lts* $lts_{targetPc}$ (Fig. 21) obtained after the occurrences of labels e_1 have been set to τ.

8.2.3 Rewriting

The set of labels of the two *lts* lts_{spt} and lts_{Pct} obtained after

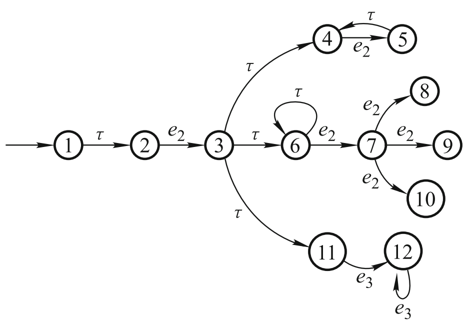

Fig. 22 Irrelevant actions set to τ in *lts* for a PC platform

removing irrelevant actions are still different sets. A rewriting step, exploiting relations on labels borrowed from the ontology is required to obtain two labelled state transition systems with a single set of labels. Let E and E' be the two sets of labels of lts_{spt} and lts_{Pct} respectively. Let $LabDiff$ be the set of different labels of lts_{spt} and lts_{Pct}. It is defined by

$$LabDiff = (E \cup E') - (E \cap E') = \{e_2, e_3, e_4\}.$$

Let E_s be the set of labels of lts_{spt} to be rewritten.

$$E_s = E \cap LabDiff = \{e_4\}.$$

Let E'_p be the set of labels of lts_{Pct} to be rewritten.

$$E'_p = E' \cap LabDiff = \{e_2, e_3\}.$$

The relation Γ (see Definition 11) provided by our ontology is the equivalence and/or subsumption relationships. These relations mean that user actions and/or user interactions with the same effect (equivalent or subsumed effect) can be replaced by the interactive task corresponding to this effect.

The result of this substitution operation is a singleton $A = \{g\}$ containing a label neither in E nor in E'. This label must be rewritten according to the identified ontological relation in order to get a same set of labels. Figure 23 shows instances of the ontology we have used to define such relations. When applied, the rewriting function ϕ (see Definition 12) produces the labels defined in Table 4.

Table 4 Label rewriting table

User actions	*lts*'s labels	Substitution action	ϕ Application
(Click LBTN, Point Scrn)	(e_2, e_4)	GO	$\phi(e_2, e_4) = g$
(Click RBTN, Point Scrn)	(e_3, e_4)	GO	$\phi(e_2, e_4) = g$

The label g, introduced in Table 4, corresponds to the interactive task GO. It defines a substitute for the labels corresponding to the user actions Click and Point (see Table 3). Indeed, this label factorizes and represents the effect of the two actions.

After rewriting the labels of lts_{spt} and lts_{Pct}, the new labelled state transition systems lts_{sptr} and lts_{Pctr} are obtained.

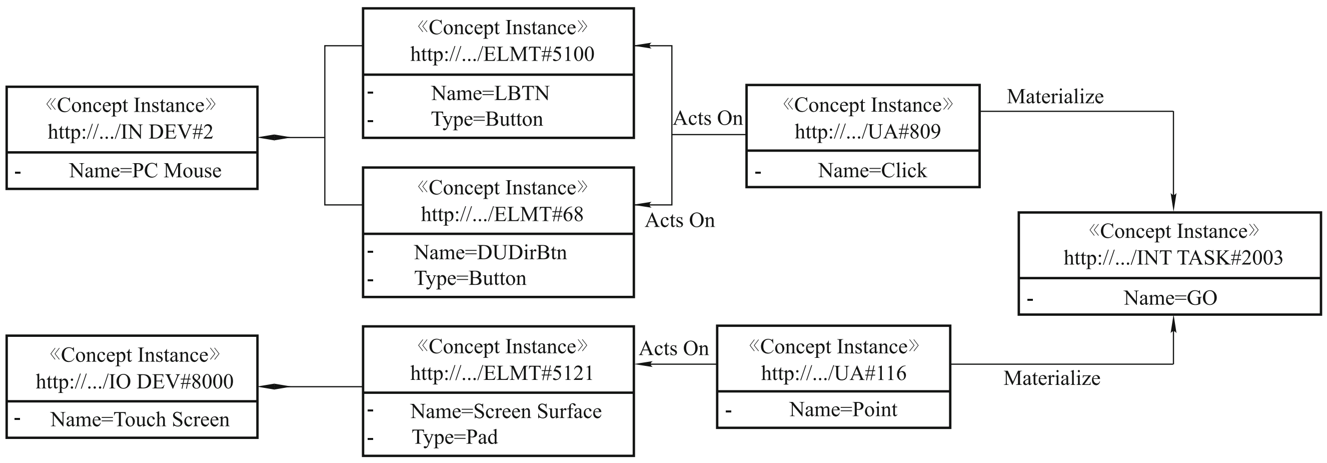

Fig. 23 Instances of our ontology concepts used to rewrite labels

Figure 24 shows the obtained *lts* after rewriting for a smartphone platform.

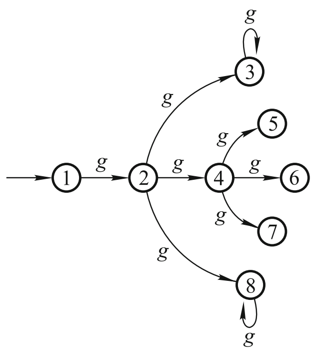

Fig. 24 The TryStage *lts* for a smartphone platform after rewriting labels

Figure 25 shows the obtained *lts* after rewriting for a personal computer platform.

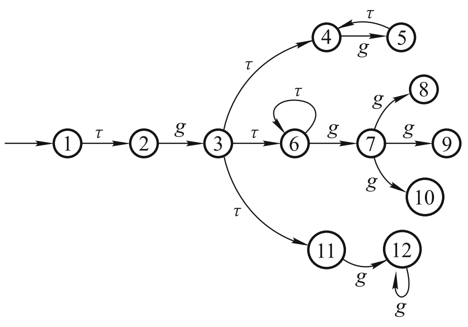

Fig. 25 The TryStage *lts* for a PC platform after rewriting labels

After this rewriting step, the two *lts* have the same set of labels which is $\{g, \tau\}$.

8.2.4 Checking

As for the previous case study, the final step concerns checking the behavior equivalence on the obtained labelled state transition systems. In the same manner, observational equivalence between the two obtained labelled state transition systems is checked. The final result shows that lts_{spctr} and lts_{Pctr} are weakly bi-similar. Therefore, we can formally assert that the interactions described by the task models of Figs. 18 and 19 perform the same task, and thus the devices and the interactions may be substituted. In other words, the equivalence of the corresponding rewritten labelled state transition systems modulo the *relational weak bi-simulation relationship* means that these two interaction techniques can substitute each other.

Like in the previous case study, the CADP model checker [60] is used to compare labelled state transition systems. Thus, the labelled state transition systems of our second case study are first described within in LotosNt [60] (see Fig. 26), then they are transformed to full Lotos programs. Labelled sate-transitions systems in BCG format are generated for each Lotos description. Finally the two BCG automata are compared with the bi-simulator modulo observational equivalence relation.

9 Conclusion

This paper presents a formal approach to check the plasticity property of user interfaces. Our work helps user interface designers to find a suitable alternative when interaction devices, of a user interface running platform, change.

9.1 Obtained results

We have proposed an approach supporting the verification of the equivalence of task models designed to describe two interaction techniques to achieve the same task with different interaction devices. The abstract part (representing the abstract task independently from any platform, and any environment) remains the same but several implementations are possible depending on the interaction devices (input/output) within the platform where the system or its interface (case of distributed systems) is running (concrete task). We started from a task model [21] representing the user task to carry out within the system and then we extracted the underlying labelled state transition system.

```
module mrblegsp is
process main [g: any] ( ) is
   g;
   select
     g;  mrblsp1 [g]
   [ ]
     g;  mrblsp2 [g]
   [ ]
     g;  mrblsp3 [g]
   end select
end process -- main
process mrblsp1 [g: any] ( ) is
      g;
      mrblsp1 [g]
end process -- mrblsp1: part 1
 ...
 ...
process mrblsp3 [g: any] ( ) is
      g;
      mrblsp3 [g]
end process -- mrblsp3: part 3
end module -- mrblegsp
```

Fig. 26 A section of LotosNt code corresponding to lts_{spt}

We have exploited checking of the equivalence of heterogeneous labelled state transition systems or *lts* with different sets of labels defined in [61]. Classical weak bi-simulation is extended by the use of an explicit relation to link labels of *lts* so that these *lts* can be rewritten to *lts* with the same set of labels and compared modulo weak bi-simulation. The defined approach has been applied to compare task models representing several interaction techniques in the field of plastic user interfaces. A domain ontology of interaction techniques and devices has been proposed to provide the relation which links labels of the *lts* representing task models at interaction level.

The application of our approach has been illustrated on two case studies through which we have shown how to check with formal tools if a task model designed for an application on a personal computer platform is equivalent to the task model designed for the same application but for another platform, a smartphone, a Touch Pad or a PC. This approach is particularly useful, for instance, to compare design strategies to face input/output hardware failure in critical interactive systems.

9.2 Discussion

The approach we have proposed relies on two pillars. The first one is the use of an ontology to model domain properties. These properties are made explicit in the formalized task models. The second one relates to the use of formal methods to check behavior equivalence. Weak bi-simulation is used for this purpose. The interest of the developed approach is the separation of concerns. Indeed, the ontology of interaction may evolve independently of the task models definition without altering the defined approach for checking user interfaces plasticity. The proposed approach is modular, it uses the ontology to define relations on labels. The ontology may evolve independently of any application in order to integrate new devices and/or interaction modes.

However, the work we have presented still require some methodological improvements out of the scope of this paper.

- The first one consists requiring the existence of an agreed, shared and consensual ontology for the human-computer interface domain which plays the role of a shared standard. This ontology, if available, shall describe unambiguously, interaction devices, interaction modes, and basic tasks, etc. Standardization bodies or UI designers communities can define and manage such ontologies.
- Then, like for the devices and interaction modes, the rewriting of labels obeys to domain-specific rules that shall be expressed in this ontology. A designer may be able to identify equivalences or subsumptions between the concepts of the human computer interaction devices so that he/she can select which label may replace a given one.
- The use of model checking and exhaustive state exploration for the verification of weak bi-simulation may lead to the explosion of the number of explored states. The complexity of the approach relies on the complexity of the weak bi-simulation checking and ontology reasoning algorithms. One may observe that the approach we have developed does not require model checking as unique verification procedure. Scalable techniques, like proof based methods, can be set up to handle this verification.

9.3 Some perspectives

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 [62] or within logic reasoners like Racer [63] or Pellet [64]. 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.

Moreover, we also plan to study the case of degradation of an interactive system and use this approach for process adaptation of plastic user interfaces. Furthermore, the application of this approach in software adaptation and the comparison of web services composition or orchestration strategies can be envisaged.

References

1. Thevenin D, Coutaz J. Plasticity of user interfaces: framework and research agenda. In: Proceedings of INTERACT. 1999, 110–117
2. Coutaz J, Calvary G. HCI and software engineering for user interface plasticity. In: Jacko J A, ed. HCI Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, 3rd ed. Boca Raton, FL: CRC Press, 2012, 1195–1220
3. Navarre D, Palanque P, Basnyat S. A formal approach for user interaction reconfiguration of safety critical interactive systems. In: Proceedings of International Conference on Computer Safety, Reliability, and Security. 2008, 373–386
4. Calvary G, Coutaz J, Bouillon L, Florins M, Limbourg Q, Marucci L, Paternò F, Santoro C, Souchon N, Thevenin D, Vanderdonckt J. The cameleon reference framework. Deliverable D1 of the Cameleon project, 2002
5. Mori G, Paternò F, Santoro C. Tool support for designing nomadic applications. In: Proceedings of the 8th International Conference on Intelligent User Interfaces. 2003, 141–148
6. Samaan K. Prise en Compte du Modèle d'Interaction dans le Processus de Construction et d'Adaptation d'Applications Interactives. Dissertation for the Doctoral Degree. Lyon: Ecole Centrale de Lyon, 2006
7. Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López-Jaquero V. Usixml: a language supporting multi-path development of user interfaces. Engineering HCI and Interactive Systems, 2005, 134–135
8. Palanque P, Paternò F. Formal Methods in Human-Computer Interaction. New York: Springer-Verlag, 1997
9. Hartson H R, Siochi A C, Hix D. The UAN: a user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems (TOIS), 1990, 8(3): 181–203
10. Rix D, Hartson H. Developping User Interfaces: ensuring Usability Through Product & Process. New York: John Wiley & Sons, inc., 1993
11. Dix A, Finlay J, Abowd G, Beale R. Human-Computer Interaction. Upper Saddle River: Prentice Hall, 1993
12. Paternò F, Mancini C, Meniconi S. Concurtasktrees: a diagrammatic notation for specifying task models. In: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction. 1997, 362–369
13. Paternò F, Santoro C. Integrating model checking and HCI tools to help designers verify user interface properties. In: Palanque P, Paternò F, eds. Interactive Systems Design, Specification, and Verification. Lecture Notes in Computer Science, Vol 1946. Berlin: Springer, 2001, 135–150
14. Scapin D, Pierret-Golbreich C. Towards a method for task description: MAD. Work with Display Units, 1989, 89: 371–380
15. Scapin D, Bastien J. Analyse des tâches et aide ergonomique à la conception: l'approche mad*. Analyse et conception de l'IHM, 2001, 85–116
16. Sybille C, Dominique S, Patrick G, Mickael B, Francis J. Increasing the expressive power of task analysis: systematic comparison and empirical assessment of tool-supported task models. Interacting with Computers, 2010
17. Chebieb K, Mansour D, Ait-Ameur Y. Analyse et evaluation de propriétés dans les ihm. In: Proceedings of the 7th International Symposium on Programming and Systems. 2001, 241–252
18. Ait Ameur Y, Kamel N. A generic formal specification of fusion of modalities in a multimodal HCI. Building the Information Society, 2004, 415–420
19. Palanque P, Bastide R. Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In: Paternó F, eds. Design, Specification and Verification of Interactive Systems. Focus on Computer Graphics. Berlin: Springer, 1994, 383–400
20. Navarre D, Palanque P, Ladry J F, Barboni E. ICOs: a model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Transactions on Computer-Human Interaction, 2009, 16(4): 18
21. Paternó F, Mori G, Galiberti R. CTTE: an environment for analysis and development of task models of cooperative applications. In: Proceedings of CHI '01 Extended Abstracts on Human Factors in Computing Systems. 2001, 21–22
22. Ait Ameur Y, Baron M, Kamel N, Mota J M. Encoding a process algebra using the event B method: application to the validation of human-computer interactions. International Journal on Software Tools for Technology Transfer, 2009, 11(3): 239–253
23. Mohand-Oussaïd L, Aït-Sadoune I, Aït-Ameur Y. Modelling information fission in output multi-modal interactive systems using event-B. In: Proceedings of the 1st International Conference on Model and Data Engineering. 2011, 200–213
24. Duke D, Harrison M D. Event model of human-system interaction. IEEE Software Engineering Journal, 1995, 10(1): 3–10
25. Brun P. XTL: a temporal logic for the formal development of interactive systems. Formal Methods for Human-Computer Interaction, 1997, 121–139
26. D'Ausbourg B. Using model checking for the automatic validation of user interface systems. In: Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems. 1998, 242–260
27. Ait Ameur Y, Kamel N. A generic formal specification of fusion of modalities in a multimodal HCI. In: Jacquart R, eds. IFIP World Computer Science. 2004, 415–420

28. Milner R. A Calculus of Communicating Systems. Secaucus, NJ: Springer-Verlag New York, Inc., 1982
29. Lotos I S O. A formal description technique based on the temporal ordering of observational behaviour. International Organisation for Standardization-Information Processing Systems — Open Systems Interconnection, Geneva, 1988
30. Dictionary C. Cambridge Dictionaries Online, 2002
31. Vanderdonckt J, Grolaux D, Van Roy P, Limbourg Q, Macq B, Michel B. A design space for context-sensitive user interfaces. In: Proceedings of IASSE, 2005, 207–214
32. Johnson J A, Nardi B A, Zarmer C L, Miller J R. ACE: building interactive graphical applications. Communications of the ACM, 1993, 36(4): 40–55
33. Kawai S, Aida H, Saito T. Designing interface toolkit with dynamic selectable modality. In: Proceedings of the 2nd Annual ACM Conference on Assistive Technologies. 1996, 72–79
34. Crease M. A toolkit of resource-sensitive, multimodal widgets. University of Glasgow, 2001
35. Bier E A, Stone M C, Pier K, Buxton W, DeRose T D. Toolglass and magic lenses: the see-through interface. In: Proceedings of the 20th Annual Conference on CGIT. 1993, 73–80
36. Stuerzlinger W, Chapuis O, Phillips D, Roussel N. User interface façades: towards fully adaptable user interfaces. In: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology. 2006, 309–318
37. Demeure A, Calvary G, Coninx K. Comet(s), a software architecture style and an interactors toolkit for plastic user interfaces. In: Graham T, Palanque P, eds. Interactive Systems. Design, Specification, and Verification. Lecture Notes in Computer Sciences, Vol 5136. 2008, 225–237
38. Jabarin B, Graham T C M. Architectures for widget-level plasticity. In: Proceedings of International Workshop on Interactive Systems. Design, Specification, and Verification of Interactive Systems. 2003, 451–460
39. Stanciulescu A. Methodology for Developing Multimodal User Interfaces of Information Systems. Leuven: Presses univ. de Louvain, 2008
40. Samaan K, Tarpin-Bernard F. The AMF architecture in a multiple user interface generation process. In: Proceedings of Developing User Interfaces with XML. 2004
41. Dery-Pinna A M, Fierstone J, Picard E. Component model and programming: a first step to manage human computer interaction adaptation. In: Proceedings of International Conference on Human-Computer Interaction with Mobile Devices and Services. 2003, 456–460
42. De Oliveira K M, Bacha F, Mnasser H, Abed M. Transportation ontology definition and application for the content personalization of user interfaces. Expert Systems with Applications, 2013, 40(8): 3145–3159
43. Sonnenberg J. 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. 2010, 162–165
44. Dees W. Usability of nomadic user interfaces. In: Jacko J, eds. Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments Lecture Notes in Computer Science, Vol 6763. Berlin: Springer, 2011, 195–204
45. Masson D, Demeure A, Calvary G. Examples galleries generated by interactive genetic algorithms. In: Procedings of the 2nd Conference on Creativity and Innovation in Design. 2011, 61–71
46. Pierre D, Marc D, Philippe R. Ubiquitous widgets: Designing interactions architecture for adaptive mobile applications. In: Proceedings of International Conference on Distributed Computing in Sensor Systems. 2013, 331–336
47. Demeure A. Modèles et outils pour la conception et l'exécution d'Interfaces Homme-Machine Plastiques. Dissertation for the Doctoral Degree. Grenoble: Université Joseph Fourier, 2007
48. Gruber T R. A translation approach to portable ontology specifications. Knowledge Acquisition, 1993, 5(2): 199–220
49. Jean S, Pierra G, Ait Ameur Y. Domain ontologies: a database-oriented analysis. In: Filipe J, Cordeiro J, Pedrosa V, eds. Web Information Systems and Technologies. Lecture Notes in Business Information Processing, Vol 1. Berlin: Springer, 2007, 238–254
50. Demeure A, Calvary G. Le modèle d'évolution en plasticité des interfaces: apport des graphes conceptuels. In: Actes de la 15ème conférence francophone IHM 2003. 2003, 80–87
51. Rekimoto J. Pick-and-drop: a direct manipulation technique for multiple computer environments. In: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology. 1997, 31–39
52. Constantine L L. Canonical abstract prototypes for abstract visual and interaction design. In: Proceedings of International Workshop on Design, Specification, and Verification of Interactive Systems. 2003, 1–15
53. Milner R. Communication and Concurrency. Upper Saddle River, NJ: Prentice-Hall, Inc., 1989
54. Chebieb A, Ait-Ameur Y. Formal verification of plastic user interfaces exploiting domain ontologies. In: Proceedings of the 9th International Symposium on Theoretical Aspects of Software Engineering. 2015, 79–86
55. Ait Ameur Y, Ait Sadoune I, Mota J M, Baron M. Validation et vérification formelles de systèmes interactifs multi-modaux fondées sur la preuve. In: Proceedings of the 18th International Conference of the Association Francophone d'Interaction Homme-Machine. 2006, 123–130
56. Buxton W. A three-state model of graphical input. In: Proceeding of Human-Computer Interaction-INTERACT. 1990, 449–456
57. Card S, Mackinlay J D, Robertson G. The design space of input devices. In: Proceedings of the ACM Conference on Human Factors in Computing Systems, Multi-Media. 1990, 117–124
58. Frohlich D. The design space of interfaces. In: Proceeding of the 1st Eurographics Workshop on Multimedia Systems, Interaction and Applications. 1991
59. Dragicevic P, Fekete J D. Support for input adaptability in the icon toolkit. In: Proceedings of the 6th International Conference on Multimodal Interfaces. 2004, 212–219
60. CADP-Team. http://cadp.inria.fr, 2013
61. Ait-Ameur Y, Chebieb A. A formal model to check systems substitutability: an application to interactive systems. Technical Report. 2013
62. Clavel M, Duràn F, Eker S, Lincoln P, Martì-Oliet N, Meseguer J, Quesada J. Maude: specification and programming in rewriting logic. Theoretical Computer Science, 2002, 285(2): 187–243
63. Haarslev V, Möller R. Description of the RACER system and its applications. Description Logics, 2001, 49
64. Bijan S E P. Pellet: an owl DL reasoner. In: Proceedings of International Workshop on Description Logics. 2004, 6–8

Abdelkrim Chebieb is an assistant professor and PhD candidate at Computer Science School for Engineers (ES), Algeria. He got his MS and BS in computer science (Hardware and Software systems) at the Mouloud Mammeri University of Tizi-Ouzou, Algeria.

Yamine Ait Ameur is a professor at the Polytechnique National Institute in Toulouse (INPT-ENSEEIHT), France. He is a member of the ACADIE research team at IRIT Computer Science Research Institute in Toulouse, France. Formal modeling is in the heart of his research activities. Formal methods in particular refinement and proof based methods and ontology based modeling are his main topics of interest.