Model-based Engineering of Feedforward Usability Function for GUI Widgets

Peer-reviewed author version

Navarre, Davis; Palanque, Philippe; COPPERS, Sven; LUYTEN, Kris & VANACKEN, Davy (2021) Model-based Engineering of Feedforward Usability Function for GUI Widgets. In: INTERACTING WITH COMPUTERS, 33 (1) , p. 73 -91.

DOI: 10.1093/iwcomp/iwab014
Handle: http://hdl.handle.net/1942/34124

Model-based Engineering of Feedforward for GUI Widgets

David Navarre, Philippe Palanque
ICS-IRIT, Université Toulouse III – Paul Sabatier, France
{firstname.lastname}@irit.fr

Sven Coppers, Kris Luyten, Davy Vanacken
Hasselt University - tUL - Flanders Make, Expertise Centre for Digital Media, Belgium
{firstname.lastname}@uhasselt.be

Abstract. Feedback and feedforward are two fundamental mechanisms that support users' activities while interacting with computing devices. While feedback can be easily solved by providing information to the users following the triggering of an action, feedforward is much more complex as it must provide information before an action is performed. For interactive applications where making a mistake has more impact than just reduced user comfort, correct feedforward is an essential step toward correctly informed, and thus safe, usage. Our approach, Fortunettes, is a generic mechanism providing a systematic way of designing feedforward addressing both action and presentation problems. Including a feedforward mechanism significantly increases the complexity of the interactive application hardening developers' tasks to detect and correct defects. We build upon an existing formal notation based on Petri Nets for describing the behavior of interactive applications and present an approach that allows for adding correct and consistent feedforward.

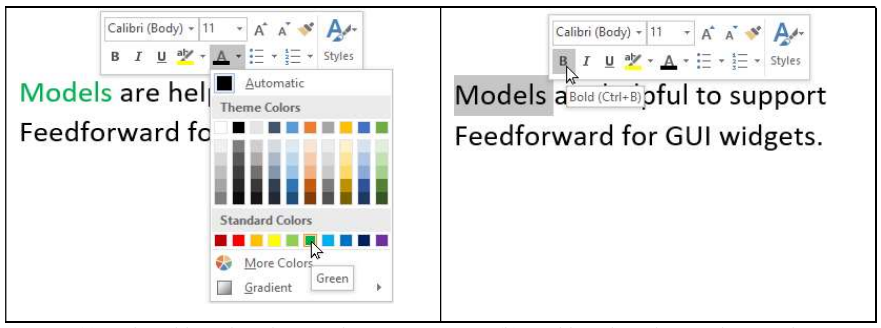
Keywords: Feedforward, formal methods, Petri nets, interactive systems engineering.

1 Introduction

As applications are becoming increasingly more complex, it becomes harder to design user interfaces that are easy to understand. According to Norman's action theory (Norman 1988), Feedback and feedforward are two fundamental mechanisms during interaction that help cross the Gulf of Evaluation (Lee & Yamada 2010) and the Gulf of Execution (Schwarz et al. 2011) respectively. While feedback explains to the user what has happened as a result of an action, feedforward provides this information beforehand, before the action is performed. Feedforward is particularly useful when the user is deciding which action to perform next or to assess the impact of the next action. Similar to the pending concept of a **security function** (Yoon et al. 2015) or a **safety function** (Lee & Yamada 2010), we argue that feedforward is a **usability function**. While a safety function can be defined as a function added to a system to prevent un-

desired safety problems (for instance a safety belt in a car does not impact driving capabilities of the driver but only improves safety), we would define a usability function as a function added to an interactive system to prevent undesired usability problems and to globally improve usability (without altering the functionalities offered by the system). Within this context, feedforward can be considered as a usability function similar to "undo". Undo is known to be difficult to implement (Zhang & Wang 2000) due to its crosscutting nature (Bass et al. 2020) which is the main reason why it is not always available in interactive applications

Similarly, despite the clear benefits, there is a lack of support for feedforward in existing GUI toolkits. There is no standardized way to specify and implement feedforward. As a result, developers often need to implement ad hoc solutions, which can lead to inconsistent feedforward behavior. In Microsoft Word (Office 2016) for example, when some text is selected, hovering over markup options such as text color will result in a preview (**Fig. 1** a), while feedforward for other typesetting options such as bold is limited to a tooltip (**Fig. 1** b).



(a) Both tooltip and markup preview (b) Tooltip only and no markup preview

Fig. 1. Inconsistent feedforward in Microsoft Word (Office 2016): (a) when hovering a color, a markup preview is rendered (text is shown green) in addition to the tooltip, (b) while hovering the 'bold' option, only a tooltip is provided and a markup preview is missing (text is not shown in bold).

Such inconsistent feedforward demonstrates the need to support both designers and developers who need to provide that usability function. On the design side, this means providing a systematic way to support how to present feedforward (to let users know what the outcome of an action will be without committing to that action) and how to interact with feedforward functions. On the development side, this means providing a systematic way to support how to implement feedforward (e.g. providing developpers with implementation patterns (such as Model-View-Controller (Buschmann 1996) for managing multiple views on the same data or object-oriented patterns for undo (Berlage 1994)). One can identify two main types of feedforward: automatic and user-triggered. Automatic feedforward is often available in well-designed interfaces and corresponds to the enabling and the disabling of user interface widget answering the question: "which functions are available?". User-triggered feedforward provides localized, contextual information to the users related to the actions that they envision triggering (e.g.

part a) of **Fig. 1**). User-triggered feedforward is usually not available in user interfaces, as it requires computing the future state of the application (if a given action is performed) and presenting this future state on the UI.

In this paper, we significantly improve previous work from Coppers et al., which presented both a generic design for feedforward and its user evaluation in terms of user experience and usability (Coppers et al. 2019). Indeed, the paper proposes a formal approach for both specification and implementation of feedforward in a systematic way. We present how high-level Petri nets such as ICOs (Navarre et al. 2009) can describe feedforward and how the resulting models are amenable to verification (to remove development faults and to identify and check properties on the interactive system offering feedforward). When WIMP interactive systems are developed following an ICO-based approach, an ICO model (i.e. a high-level Petri net) is produced that describes in a complete and unambiguous way the behavior, the interaction and the presentation for each window (Navarre et al. 2009). In this paper, we propose to produce a Petri net model (called *Feedforward net*) in addition to the model describing the application. To this end we propose a generic behavioral pattern for feedforward that eases the production of the Feedforward net, that guarantees that feedforward will always exhibit the same behavior and prevent the addition of development faults. Lastly, we present how this pattern is applied on the formal model of the initial application refining it in a systematic way to provide feedforward functionality, thus reducing development cost of this usability function.

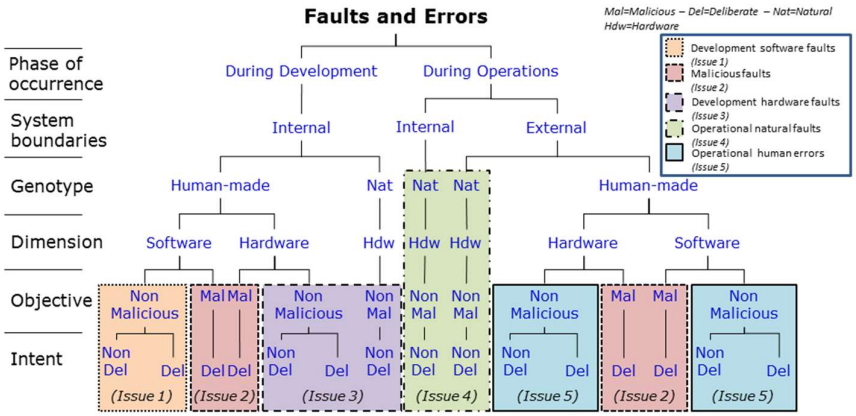


Fig. 2. Taxonomy of faults in computing systems (adapted from (Avizienis et al. 2004) and associated issues for the dependability of these systems.

To be able to ensure that the system will behave properly whatever happens, a system designer has to consider all the issues that can impair the functioning of that system. To this end the domain of dependable computing e.g. Avizienis et al. (Avizienis et al. 2004) have defined a taxonomy of faults. This taxonomy leads to the identification of 31 elementary classes of faults. **Fig. 2** presents a simplified view of this taxonomy and makes explicit the two main categories of faults (top level of the figure): i) the ones made at development time (see left-hand side of the figure) including bad designs, programming

errors, … and ii) the one made at operation times (see right-hand side of the figure) including operator errors such as slips, lapses and mistakes as defined in (Reason 1990).

The leaves of the taxonomy are grouped into five different categories as each of them bring a special problem (issue) to be addressed:

- *Development software faults (issue 1):* software faults introduced by a human during system development. They can be, for instance, bad design errors, bugs due to faulty coding, development mistakes …
- *Malicious faults (issue 2):* faults introduced by human with the deliberate objective of damaging the system. They can be, for instance, an external hack causing service denial or crash of the system.
- *Development hardware faults (issue 3):* natural (e.g. caused by a natural phenomenon without human involvement) as well as human-made faults affecting the hardware during its development. They can be, for instance, a short circuit within a processor (due to bad construction).
- *Operational natural faults (issue 4):* faults caused by a natural phenomenon without human participation, affecting hardware as well as information stored on hardware and occurring during the service of the system. As they affect hardware faults are likely to propagate to software as well. They can be, for instance, a memory alteration due to a cosmic radiation.
- *Operational human-errors (issue 5):* faults resulting from human action during the use of the system. They include faults affecting the hardware and the software, being deliberate or non-deliberate but don't encompass malicious ones. Connection between this taxonomy and classical human error classification as the one defined in (Reason 1990) can be easily made with deliberate faults corresponding to mistakes or violations (Polet et al. 2002) and non-deliberate ones being either slips or lapse.

While such taxonomy has been used in other contexts to identify dependable mechanisms for interactive systems (Fayollas et al. 2017), we use it to make explicit the link between usability functions and development faults. Indeed, the more designers add usability function to prevent human made errors (issue 5 of **Fig. 2**) the more complex the system to build and the more likely developers will add development faults (issue 1 of **Fig. 2**). This paper proposes a UI design to support users interacting with the usability function Feedforward as well as a formal design pattern for feedforward to support developers' activities (knowing what to implement , implementing always the same behavior and avoiding development faults) when implementing that function.

The remainder of this paper is structured as follows. Section 2 presents the foundations, interaction and one design for the Fortunettes concept for feedforward usability function. Section 3 presents the illustrative example of a simple widget-based interactive application that is used throughout the paper. Section 4 presents the Petri nets based modeling approach for modeling interactive applications and its application to the modelling of Fortunettes usability function. Section 5 focusses on the formal analysis of the application model and of the feedforward nets. Section 6 concludes the paper and highlight paths for future work.

2 Fortunettes: Design, Foundations and Use

2.1 Interaction

In previous work, the concept of Fortunettes was introduced (Coppers et al. 2019) as follows: a structured and precisely defined approach to integrate feedforward information about the future state of an application into standard GUI widget sets. In their user study with 104 participants, Coppers et al. found that the feedforward provided by Fortunettes helps users to understand what the outcome of an action will be, before they have performed that action.
Similar to feedback, feedforward does not need to be presented permanently. Instead, it should only be presented when more confidence is needed (Coppers et al. 2019) to avoid cluttering the UI and to prevent visual overload. We identified three situations where users are interested in feedforward information:

- when a high impact decision needs to be made (i.e. the user performs an action that is not reversible; there is no undo available for that action),
- when the user interacts with lesser known parts of the user interface,
- or when the user explicitly asks for additional information on what might happen when an action is executed.

Based on these three situations, we introduce a three-step interaction pattern to explore the future state of an application (see **Fig. 3**): (1) peek into the future, (2) go to the future, and (3) return to the present.

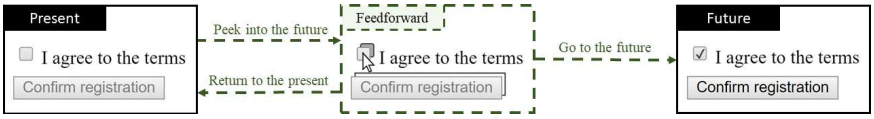


Fig. 3. Feedforward is presented in an intermediate state between the present and the future. Adapted from (Coppers et al. 2019).

When the user is considering performing an action, s/he can **peek into the future** to run a simulation of what the outcome of that action will be. The user interface shows previews about this information and allows for one of two possible actions: the user is no longer considering an action and **returns to the present**, or the user actually confirms the action, in which case the action is actually executed and the application transitions **to the future**.

In the remainder of this paper, we use mouse enter events (hover) to activate the feedforward layer and to peek into the future for all widgets, and we use mouse leave events to hide the feedforward layer and to return to the present. However, the feedforward mechanism can be adapted to become less intrusive by only showing feedforward for actions that are less likely to be understood by a user, or by choosing a different trigger to activate feedforward such as long hover. The feedforward mechanism could

also be optional (i.e. disabled by default), and only activated on-demand by holding `$CTRL$' on the keyboard, for example.

2.2 Visualization

The initial goal of this work is to inform users early about the outcome of an action. This outcome can be represented by a new application state, which consists of a set of new widgets states. We decided to embed information about the future widget states in the widgets themselves for high-cohesion, for scalability and for reusability in any widget-based application. **Fig. 4** demonstrates how standard widgets look when the design language of Fortunettes is applied to them. The design language embeds an additional feedforward component behind the widget, without interfering with existing design conventions. In this design, the feedforward component presents information about the future availability of the widget and the future value. The border of the feedforward components shows whether the widget will become disabled (dashed) or will become enabled (solid) (**Fig. 4**a). The text content and background color present the future value of the widget (**Fig. 4**b).

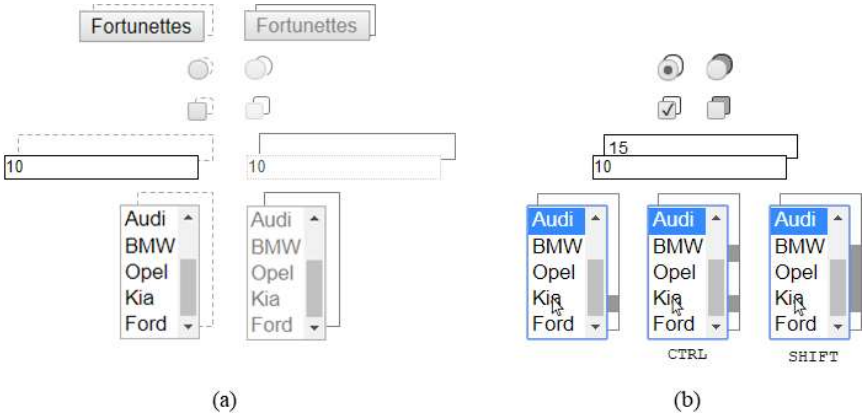


Fig. 4. A demonstration of what widgets looks like during the intermediary feedforward phase using the Fortunettes design language. The layer stacked behind the original widget provides information about the future availability and value of a widget. (a, left) Dashed borders means the widget will become disabled, whereas (a, right) a solid border means the widget will become enabled. (b) The background of the stacked layer represents the future value (Coppers et al. 2019).

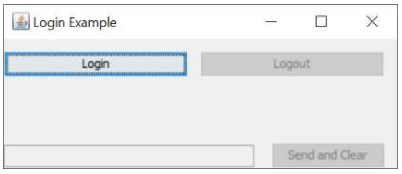
The Fortunettes design language for widgets is intended as a proof-of-concept for our feedforward mechanism. The design can definitely be improved to reduce visual clutter (especially in more complex applications), for example by aggregating feedforward information. However, design variations do not have any impact on the contributed engineering perspectives and are beyond the scope of this paper.

2.3 Examples

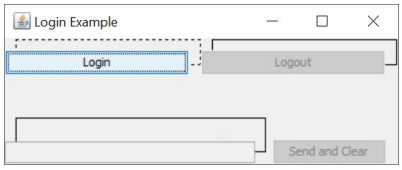
Later in this paper, we will revisit the simple login example and the more complex weather radar example presented in (Barboni et al. 2006) to discuss how they can be engineered using our novel model-based approach. For the sake of self-containment of this paper, we briefly summarize their (feedforward) behavior in the remainder of this section.

2.3.1 Login

The user interface of the login example is presented in **Fig. 5**. It is composed of four widgets (three buttons and a textbox). In **Fig. 5** a), only the '*Login*' button is enabled, and all other widgets are disabled. When a user hovers over the '*Login*' button **Fig. 5** b), the feedforward component behind each widget becomes visible to show the state that would occur when the user would click the '*Login*' button. The feedforward information explains to the user that the '*Login*' button will become disabled, and the '*Logout*' button and the textfield will become enabled. Since the state of the *'Send and Clear'* button does not change, we follow the parsimony principle of user interface designs by not showing any feedforward for this widget. Indeed, When the user clicks the 'Login' button, the interface transitions to the state depicted in **Fig. 5** c). In that state, the 'Login' and the 'Send and Clear' buttons are disabled while the 'Logout' button and the textfield are enabled. In this state, the user can write a message but cannot send when the text is empty. When the textfield is no longer empty, the interface transitions to the state depicted **Fig. 5** d), in which the *'Send and Clear'* button is enabled. If the textfield becomes empty again (by removing characters or by pressing the *'Send and Clear'* button), the application transitions back to the state depicted in **Fig. 5** c). Pressing the '*Logout*' button resets the interface to the initial state depicted in **Fig. 5** a).

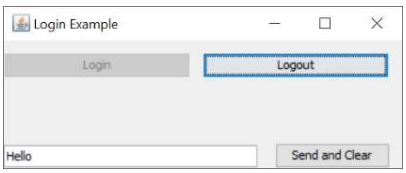


(a) The user is logged out. Only '*Login*' is enabled.



(b) When hovering '*Login*', feedforward shows that '*Logout*' and the textfield will enable, while '*Login*' will disable.

(c) When clicking '*Login*', the future state shown in (b) becomes the new state.



(d) The user has typed a message, which enables '*Send and clear*'.

Fig. 5. Screenshots of the login application.

2.3.2 Cockpit Weather Radar

The weather radar (WXR) presented in this section is built upon an application currently deployed in many cockpits of commercial aircrafts. WXR provides support to pilots' activities by increasing their awareness of meteorological phenomena during the flight journey. It allows them to determine the weather ahead of the aircraft which might end up in requesting a trajectory change, in order to avoid storms or precipitations for example. **Fig. 6** presents a screenshot of the weather radar control panel, used to operate the weather radar application. This panel provides the crew with two functionalities. The first one is dedicated to the mode selection of the weather radar and provides information about the radar status, to ensure that it can be set up correctly. The mode change can be performed in the upper part of the panel. The second functionality, available in the lower part of the window, is dedicated to the adjustment of the weather radar orientation (Tilt angle). This can be done in an automatic or a manually way (Auto/manual buttons). Additionally, a stabilization function aims at keeping the radar beam stable even in case of turbulences

Fig. 6 (left-hand side) shows the initial state of the application. In that state (lower part of the Figure) the application is in the automatic mode, i.e. the only button available is the 'Manual' one. The upper part of the Figure presents the set of five radio buttons corresponding to the five states of the weather radar. In the interaction presented WXR mode is 'Off' which is graphically made visible by the selected radio button 'Off'. The user is currently hovering the 'WXA' presenting feedforward information. If the user clicks on the 'WXA' radio button it will become selected and the 'OFF' radio button will become unselected. **Fig. 6** (center) shows the feedforward display when the user hovers the 'Manual' button from **Fig. 6** (left). In that state, feedforward information is presented for the widgets in the lower part of the window (the upper part of the window is not impacted by clicking on that button). Feedforward information on widgets informs the user that clicking on 'MANUAL' will make buttons 'AUTO' and 'OFF' available and that 'MANUAL' will become unavailable. If the user clicks on 'MANUAL' the application moves to the state represented in **Fig. 6** (right-hand side) and if the user hovers the 'OFF' button feedforward information is presented. That feedforward information tells that clicking on the 'OFF' button will make the 'ON' button and the 'TILT ANGLE' text box available. In addition, the 'OFF' button will become unavailable.

Fig. 6. Screen shots of the weather radar application that can be found in airplane cockpits (Coppers et al. 2019).

Although both user interfaces have a simple graphical interface, the underlying behavior for the weather radar is very complex. This is often the case for specialized applications that are targeted toward expert users. Even for expert users it is difficult to have a complete mental model of the behavior of the application, and some guidance may be very useful for interactions that influences the application state in a far-reaching way.

3 Modelling of Fortunettes behavior

To support the engineering of interactive applications that include feedforward, we propose an approach based on a formal description technique called Interactive Cooperative Objects (ICO). The ICO formalism is a formal description technique, based on Petri nets, dedicated to the modeling and the implementation of event-driven interfaces (Navarre et al. 2009). We firstly present in this Section the formal description technique and how it can be used to model the behavior of an interactive application, then we present how it is possible to derive the feedforward behavior of the application from the existing model of the application behavior.

3.1 ICO formal description technique

The ICO formalism uses a decomposition of communicating objects to model the system, where both behavior of objects and communication protocol between objects are described by the Petri net dialect Cooperative Objects (CO) (Navarre et al. 2009). In the ICO formalism, an object is an entity featuring four components: (1) a cooperative object which describes the behavior of the object, (2) a presentation part (i.e. the graphical interface), (3) an activation function and (4) a rendering function. These functions are the connection between the behavior of the object and the presentation (including interaction).

An ICO specification describes user interactions supported by the associated application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays state information that is relevant to the user).This formal description technique has already been applied in the field of Air Traffic Control interactive applications (Navarre et al. 2009), space command and control ground systems (Palanque et al. 2006), or military (Bastide et al. 2004) as well as civil cockpits (Barboni et al. 2006).

The ICO notation is fully supported by a CASE tool called PetShop (Bastide et al. 2002) and (Palanque et al. 2009). All the models presented in the two next Sections (3 and 5) have been edited, simulated and analyzed using PetShop tool.

3.2 Principle of Fortunettes feedforward modelling using ICO

Engineering an interactive application that includes feedforward in its representation, requires handling additional special-purpose interaction events, which we call **feedforward events**. Indeed, when the user wants to know the impact of an **action** (for instance by clicking on a button called **action**, the feedforward events are:

- an event for **peeking into the future** without changing the current application state, the name convention we use for this event is "F**action**Performed";
- an event to **go to the future** and committing to the future state that is presented and the name convention we use for this event is "InF**action**Performed";
- and an event to **return to the present** and omitting any possible future state that was shown and the name convention we use for this event is "UF**action**Performed".

These events need to be supported without affecting the behavior of the associated application. It is obvious adding feedforward can never interfere with the intended interactive behavior of an application, since its sole purpose is to show the user what will happen according to the intended behavior. Although this is evident from an interaction design point of view, it has an important impact on how feedforward behavior can be modelled. Our approach uses two core principles for modeling an application that exposes feedforward information:

1. *The feedforward behavior is modelled as an independent ICO specification, that includes a copy of the ICO model of the standard behavior.* This ensures the added feedforward fully compatible with the original application behavior. This Petri net model is called the **Feedforward net** as it allows users to look into the future of the application.
2. *Access to feedforward from within the standard behavior is modeled by forwarding the interaction to the Feedforward net when a future event occurs.* We define a special purpose pattern described in Petri nets that models the transition from standard behavior to the behavior defined in a Feedforward net.

By means of these two modelling principles, we can now exploit the behavior of the application to forecast the future states of the application if the user decides to use feedforward function.

In Fig. 7. a simple example is shown of an ICO behavioral specification for a login action. In the model, rectangles (called transitions) represent actions the system can perform while ellipses (called places) represent state variables of the system. Places can hold tokens and the distribution of tokens in the places models the current state of the model. A transition is said to be fireable if each of its input places holds at least one token. When fired, a transition removes one token from each of its input place and sets

one token in each of its output places (see (Bastide & Palanque 1999) for a detailed explanation). The login transition is the event handler for an event called loginPerformed that represents the use of the button Login. When fired, this transition moves the token from place LoggedOut (1.) to place LoggedIn, setting the state of the application to the new state following the execution of the login (corresponding code not represented here).

Fig. 7. Excerpt from the Petri net model of the standard behavior of a login dialog. In the transition, the text on the left describes the name of the transition while the text on the right (preceded by ::) describes the name of the event associated the transition.

When introducing the feedforward view on this action, three handlers for future events extra event handlers (peek into the future, go to the future and return to the present) are included. Fig. 8. illustrates this for the login example: from the event handler loginPerformed the following additional event handlers {FloginPerformed, UFloginPerformed, InFloginPerformed} are generated. The name of the generated future event handlers are generated re-using the name of the corresponding event handler, prefixed by F (that represents entering in Fortunettes mode, e.g. peek into the future), by UF (that represents exiting the Fortunettes mode, e.g. return to the present) and InF (that represents exiting the Fortunettes mode and go to the future).

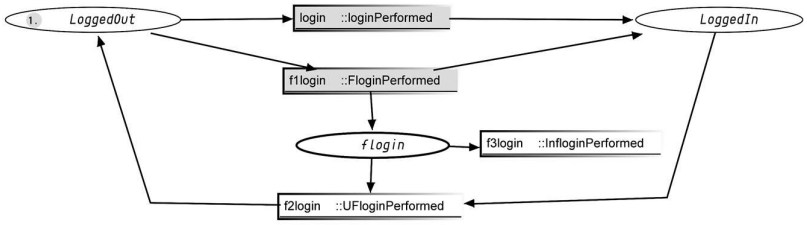


Fig. 8. Extracted from the feedforward behavior of the application: event handling of the login action and peek into its future.

In Fig. 8. , transition f1login (event handler for FloginPerformed) represents the action of peeking into the future of the action login. Basically, it behaves in the same way as the original action (put a token in place LoggedIn) while the standard behavior is still in state LoggedOut. It additionally puts a token in place flogin that represents the entering in feedforward mode (a dedicated rendering may occur).

There are then two possibilities that need to be supported by the Fortunettes pattern:

1. The user *decides to perform the action*, e.g. by clicking a button. in this case the login actionis triggered by clicking the login button, which results in the production of two events: loginPerformed handled by the standard behavior (making it going to the state LoggedIn) and InFloginPerformed handled by the feedforward

behavior (discarding the token in place flogin, while the token in place LoggedIn does not move, placing it in the same state as the standard behavior).

- The user *decides not to perform the action,* e.g. by moving the mouse away from the widget that is in focus. In this case the *login* action is not triggered by moving the mouse away from the login button, and an event UFloginPerformed. The standard behavior remains in the same state while in the feedforward behavior, the tokens from places LoggedIn and flogin are removed and a token goes back to the place LoggedOut, making it return to its previous state (leaving the feedforward mode).

This pattern is particularly efficient when describing feedforward behavior for events that do not handle values or when the widgets are simple such as button. This pattern may need to be extended to cover a wider range of (more complex) cases:

- When values are handled by the action performed by the widget, it is not always possible to peek into the future for these values. In this case, the pattern is extended with two steps: when entering the feedforward mode, an envisioned value must be produced (decided at design time for instance and presented to the user when feedforward is triggered) and when the user performs the action, a substitution must be presented between the envisioned value and the real value. In the feedforward behavior, this can be done when tokens are created. For instance, if such value was needed in the login application, the token set in place LoggedIn by transition f1login (see **Fig. 8**) would hold the design-time envisioned value, and when f3login would be fired, this token would have been removed and the correct value resulting from the performance of the action.
- When the widget is more complex (meaning it might produce multiple events) extra event handlers may need to be introduced. For instance, when using a classical textbox, one may be interested in validating the text only when the full text is entered, and not during the text input. According to the standard behavior of the application, the only event that would occur, is the last one (for instance, the event actionPerformed of the JTextField in Java Swing). On the feedforward behavior side, any text change may be relevant in order, for instance, to allow the rendering of text filtering after each key is pressed. This means that the feedforward net must exhibit a more complex behavior that has to be described at design time. This argues in favor of not automatically generating the feedforward net but to synthetize it (Badouel et al. 1995) while leaving space for tuning and refinement at design time.

Fortunettes requires enhancing widgets with extra means to allow rendering feedforward states and to trigger dedicated events. In our implementation using Java Swing widgets, we embed them within a specialized decorator, but there are many other implementation options at widget level or at application level.

3.3 Application of the modeling principle to the illustrative example

This Section presents the ICO models for both the standard application and its Fortunettes enhancement. For each model, we present the behavioral part and the two user interface description functions: the activation part and the rendering part.

Standard behavior.

Fig. 9. presents the entire behavior of the illustrative example. It may be divided into two parts: the upper part deals with login actions while the lower part deals with messages handling.

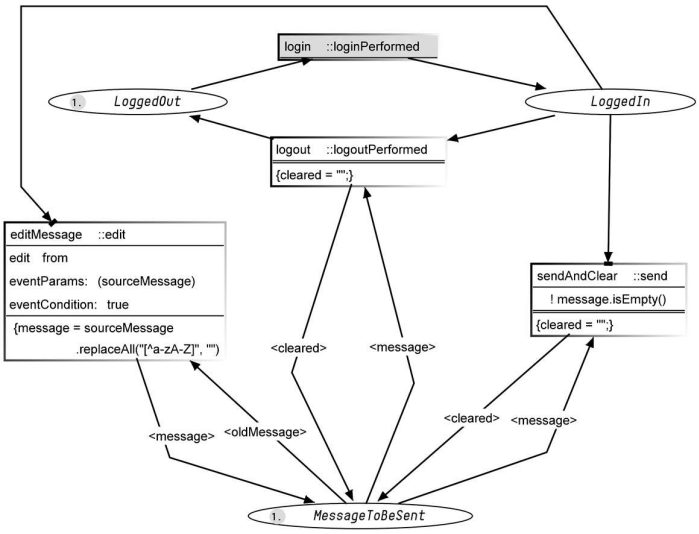


Fig. 9. Behavior of the Login example using the ICO formal description technique.

The upper part of **Fig. 9.** models what has been explained in the beginning of the Section (see **Fig. 7.**) to introduce Fortunettes and the modelling approach, including the complete behavior of the application i.e. its functional code (inside the transitions). Another difference is the way back from place LoggedIn to place LoggedOut when logging out that clears the edited message (modification of the value of the token held by place MessageToBeSent).

The lower part of **Fig. 9.** is dedicated to the message editing and to send it. Sending it (transition sendAndClear) can only occur if the message is not empty (precondition !message.isEmpty()). When it occurs, the token held by place MessageToBeSent is destroyed and a new token (with an empty string) is set to that place. The message editing is represented by the transition editMessage that receives an event called edit, and this event holds a string value called sourceMessage. This sourceMessage is then filtered resulting in a string message that only contains characters that belongs to [a-z] and [A-Z] (For instance "a1b2c3" will be transformed into "abc") by the execution of the function replaceAll.

Table 1. represents the activation function of the application. It relates the event production from the application and event handlers described using ICO. When the event occurs, the corresponding transition is fired. If the transition is not available, the corresponding event source must be disabled. This part of the functioning is assumed

by the activation rendering method (last column of **Table 1.**) that is provided by the application: for instance, setLoginEnabled changes the enabling of the button Login.

Table 1. Activation function for the ICO model of the Login example.

User Event	Event handler	Activation Rendering
Edit	editMessage	setEditEnabled
Login	login	setLoginEnabled
Logout	logout	setLogoutEnabled
Send	sendAndClear	setSendEnabled

Table 2. represents the rendering function of the application. It relates any state change within the application behavior to rendering methods call. For instance, when a token enters place MessageToBeSent, the string of this message is set in the text box widget by calling the method showMessage.

Table 2. Rendering function for the ICO model of the Login example.

ObCS node name	ObCS event	Rendering method
MessageToBeSent	marking_reset	showMessage
MessageToBeSent	token_enter	showInitialMessage

Feedforward behavior.

Fig. 10. illustrates how feedforward information can be displayed using Fortunettes. **Fig. 11** and **Table 4.** fully describe the feedforward part of the application. The behavior presented by **Fig. 11.** is structured similarly to the standard behavior (of **Fig. 9**), the upper part being dedicated to the login actions and the lower part, to the message editing.

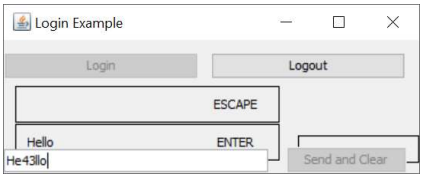


Fig. 10. Illustration of the text filtering while typing in feedforward mode

This Feedforward net behaves according to the pattern explained in the previous Section with the particularity of the filtering of the text while it is being typed in and not only at the end of the interaction with the text box (transition f4editMessage in the lower part of **Fig. 11.**). This allows to present to the user what will happen to the edited value if it is validated (e.g. press ENTER), as illustrated by **Fig. 10.**

Table 3. presents the activation of the feedforward behavior of the application. The interesting part of this function is that the activation rendering is not related to the immediate availability of the events, but to their availability in the future. Therefore, it does not directly impact the application widgets but only calls functions that have been

added to render their Fortunettes appearance. For instance, on **Fig. 10.** , if the edited text is validated (e.g. pressing ENTER), the button "Send and Clear" will become available (represented by the rectangle around it, in the background).

Table 3. Activation function for the ICO model of the feedforward behavior of the example.

User Event	Event handler	Activation Rendering
Edit	editMessage	setFortunettesEditEnabled
Login	login	setFortunettesLoginEnabled
Logout	logout	setFortunettesLogoutEnabled
Send	sendAndClear	setFortunettesSendEnabled

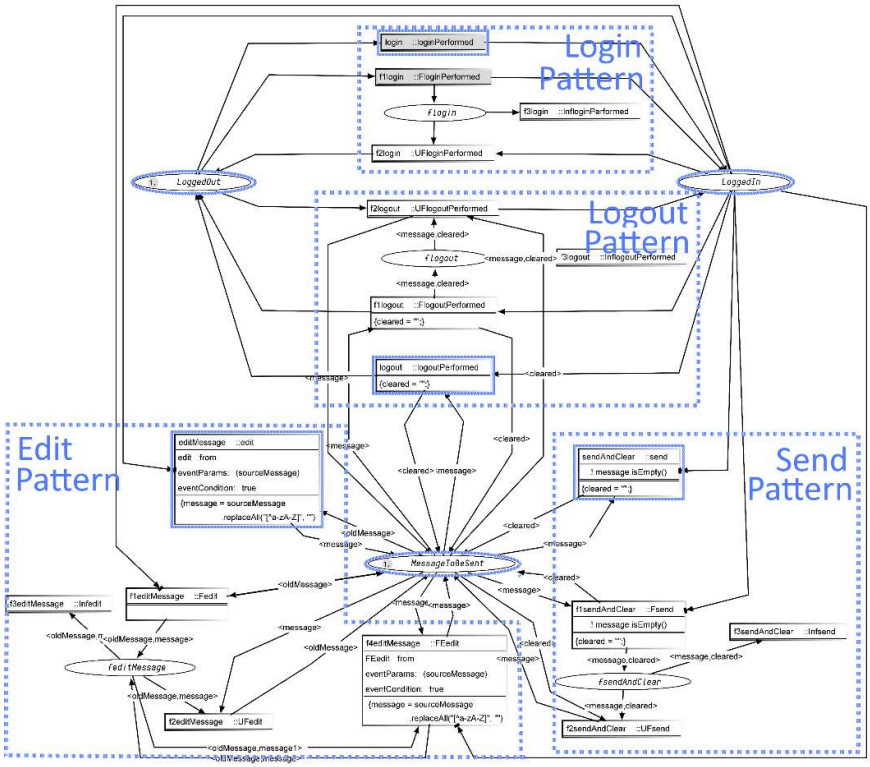


Fig. 11. The Feedforward net describing the feedforward behavior of the Login example using the ICO formal description technique.

Table 4. presents the rendering function of the feedforward behavior of the application. This function first aims at making the application entering in feedforward mode (a token enters any of the places prefixed f) or at exiting the feedforward mode (a token exits any of the paces prefixed by f). This function ensures too that when a new message is under editing, it is rendered on the feedforward part of the interface (each time a token

enters the place MessageToBeSent, showFortunettesMessage is called modifying what is rendered in the ENTER rectangle of the text box as illustrated on Fig. 10.).

Table 4. Rendering function for the ICO model of the feedforward behavior of the example.

ObCS node name	ObCS event	Rendering method
MessageToBeSent	marking_reset	showFortunettesMessage
MessageToBeSent	token_enter	showFortunettesInitialMessage
fEditMessage	token_enter	startRenderFortunettes
fEditMessage	token_exit	stopRenderFortunettes
fLogin	token_enter	startRenderFortunettes
fLogin	token_exit	stopRenderFortunettes
fLogout	token_enter	startRenderFortunettes
fLogout	token_exit	stopRenderFortunettes
fSendAndClear	token_enter	startRenderFortunettes
fSendAndClear	token_exit	stopRenderFortunettes

This joint, carefully coordinated behavior between the standard behavior of the application and its Fortunettes ones is highlighted on **Fig. 10.** Indeed, when the user types some text in, it is rendered directly in the text box while the Fortunettes rendering displays the text, as it will appear if the validation key is pressed. In the case of the login application, we see that all the non-textual characters will be removed and the current text "He43llo" will appear as "Hello" in the future.

4 A Model-Based Engineering Process for Feedforward nets

Our approach requires the creation of additional models, that would not be necessary when building an interactive system without feedforward. In this section, we present the overall process that can be followed in order to include feedforward information to support the use of safety-critical systems. Since this has important impact on the engineering process for building an interactive system, feedforward is often only included for the parts of the interactive system where users could perform an unintended sequence of interactions that lead to a failure and comes with a large cost. This means that during development time, we want to enable designers and developers to harness the user interface against "issue 5: Operational Human Errors" (see **Fig. 2**). As a side effect, considering feedforward information during the design and development stages of the engineering process helps both designers and developers to explicitly consider complex and possibly confusing situations in the user interface and empower the user to take informed actions.

The benefits of having feedforward included in an interactive system comes with the cost of extra modeling work and managing more complex models. However, our approach ensures (1) consistency, by using the same notation for the feedforward net and defining a pattern of how to generate a feedforward net, (2) independence of feedforward, by separating the feedforward net from the application behavior, and (3) strong validation during development, by allowing for formal analysis and by providing a sandbox model to execute and test the models. Furthermore, we build on top of existing,

proven tools to specify, validate and execute ICO models. We start from a standard ICO model that describes the behavior of an interactive system. A feedforward net, also using the same ICO notation, is generated based on this model. **Fig. 12** depicts the overall process. The developer can iterate over the ICO model, applying changes and corrections. These changes might imply updates that need to be propagated to the Feedforward net too. However, in our current approach we rely on the developer to decide whether it is necessary to adapt the feedforward net. Automated suggestion on how to update the Feedforward net could be useful here, and might help the developer to deal with the complex relationship between the ICO model and the Feedforward net. However, the Feedforward net is used to specify and design the selected feedforward information that needs to be included, and not all aspects of a user interface benefit from adding feedforward.

The developer can also iterate over the Feedforward net separately. This is necessary since not all aspects of an interactive application require feedforward, and some other parts of the system might require more extensive feedforward behavior. Our approach does not limit these independent adaptations in both models, however, offers a basic sanity check through the application of the aforementioned patterns.

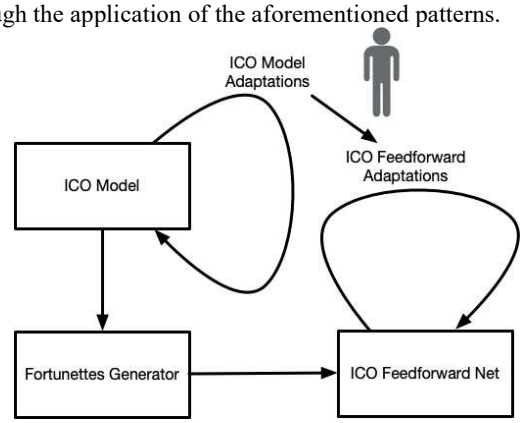


Fig. 12. The proposed process for producing ICO feedforward nets from ICO models of interactive applications

When the design and development of the application reaches a state in which it can be executed, we can test and validate it using the ICO runtime environment. **Fig. 13**. depicts how the runtime environment comes into play. User events are transmitted to both the standard ICO model as well as the ICO feedforward model. In this case a click-event triggers an action and progresses both the standard ICO model as well as the ICO feedforward model. A hover, on the other hand, triggers the ICO feedforward net thus computes and presents what the potential outcome is if one would confirm the action with a click. Notice the trigger to show feedforward, i.e. a hover action, can be replaced by any other actions supported by ICO. The Petshop environment (Bastide et al. 2002) and (Navarre et al. 2009) is used for editing and for the interpretation of the ICO models, presenting the embedded user interface and for managing user interaction. PetShop supports direct communication between the ICO models and the application logic, thus

can be used both as a development as well as a runtime environment. A screenshot of the environment being used during development time can be found in **Fig. 14**.

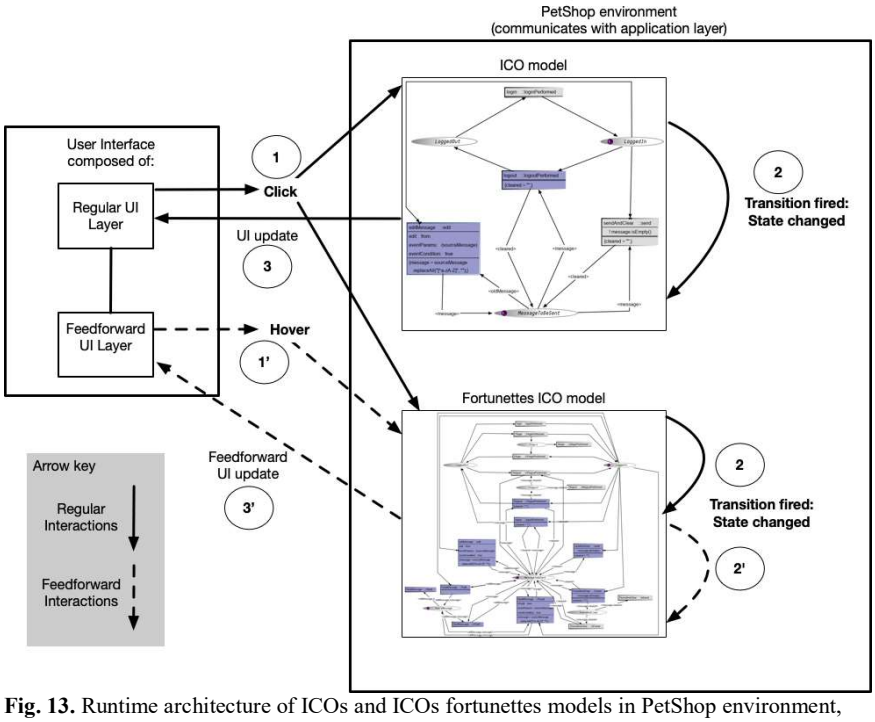


Fig. 13. Runtime architecture of ICOs and ICOs fortunettes models in PetShop environment, where a hover action is used to trigger feedforward. Sequence 1-2-3 presents what happens when an actual interaction is triggered. Sequence 1'-2'-3' presents what happens when feedforward is triggered.

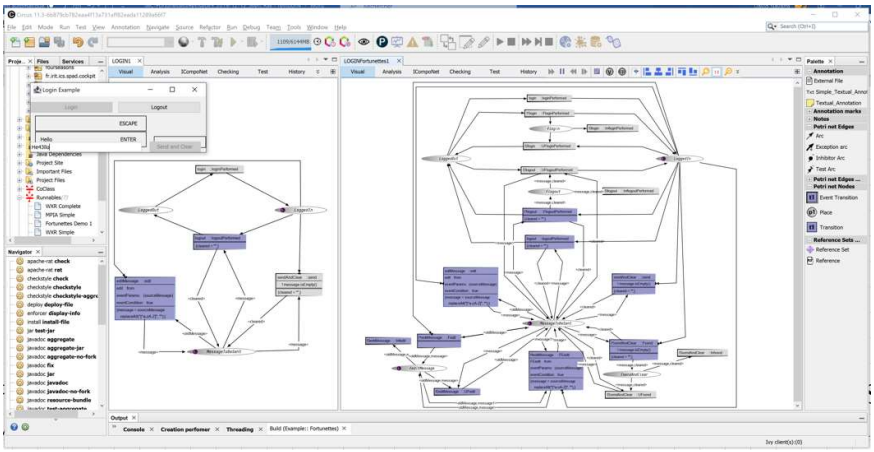


Fig. 14. The PetShop environment being used to create and validate the Fortunettes ICO model during development. The screenshot shows the standard ICO model (left) and the feedforward ICO model (right), and the associated user interface (top left).

It is important to note tht we support both the simulation and the execution of ICO models. Thus both ICO-based specification of the regular behavior as well as the feedforward model are concurrently executed at runtime: during usage of an interactive application.

5 Formal Analysis on the illustrative example

This Section is dedicated to the formal analysis of the models presented above. The fact that we produce two different models for the same application (the standard application model and the Feedforward net) has multiple implications. First, the standard application models must exhibit some properties and it is important to check that they are true. An example of such property could be that a given Button of the application remains always available. Second, the Feedforward net also needs to exhibit some properties (e.g. each time the user triggers the "peek into the future" there must be two actions available: one to go into that peeked future and one to come back to the current present. Third, the Feedforward nets must implement a "similar" behavior as the standard application and thus we must demonstrate their compatibility. For instance, it is important to demonstrate that all the actions available in the models of the standard application are also available in the Feedforward net. This is an example of the development faults that could be added by developers when the feedforward usability function is added to an application.

With ICOs, as detailed in (Silva et al. 2013) and (Palanque et al. 1995), there are two different formal analysis techniques:

- The analysis of the underlying Petri net using results from Petri nets theory. This analysis can be performed using methods and algorithms from the Petri nets community such as the ones presented in (Murata 1989).
- The analysis of the high-level Petri net (ICO) but this requires manual demonstrations as some of the properties are undecidable (Dietze et al. 2007).

Due to space constraints, we only present here properties that are based on the underlying Petri net model. Some interesting results demonstrate that the high-level nature of the Petri nets in the ICO notation only reduce the availability of transitions (for instance when they feature pre-conditions) and thus in order for the high-level Petri net to be live, the underlying Petri net must be live (Bastide et al. 1993).

5.1 Formal analysis of the model of the standard behavior (Fig. 9.)

Table 5. presents the list of traps and siphons of the model in **Fig. 9.** [1]. In a Petri net a siphon is a set of places that never gain tokens whatever transition is fired while a trap is a set of places that never lose tokens (David & Alla 1992). The fact that all the places

[1] The computation of the results in those tables was done using Petshop tool and are not presented due to space constraints. How this computation is performed is presented in (Bastide et al. 2002).

in the model are both traps and siphons demonstrate that the number of tokens in the model will remain the same as the one in the initial state i.e. two tokens (see **Fig. 9.**).

Table 5. Siphons and Traps from the standard behavior of the application.

Siphons	Traps
MessageToBeSent	MessageToBeSent
LoggedIn, LoggedOut	LoggedIn, LoggedOut

Table 6. analysis is based on the calculation of transition invariants and place invariants. As can be seen all the places in the model belong to a place invariant which means that the total number of tokens in the places of the models will remain the same. One interesting piece of information is that place MessageToBeSent is a single place in a P-invariant. This means that whatever transition is fired the number of tokens in that place will always be the same as the one of the initial marking. In the current example, this means that the place MessageToBeSent will always be marked by a single token.

Table 6. Transitions and Place Invariants from the standard behavior of the application.

T-Invariants	P-Invariants
1 sendAndClear	1 LoggedIn, 1 LoggedOut
1 editMessage	1 MessageToBeSent
1 login, 1 logout	

In terms of behavior, transitions Login and Logout belong to the same t-invariant which means that, if they can be made available from the initial state, there always exists a sequence of transitions in the Petri net to make them available again. Their connection with the P-invariant {1 LoggedIn, 1 LoggedOut} (with a bounded value of one token) demonstrates that always one of the two transition will be available and they will never be available at the same time.

5.2 Formal analysis of the Feedforward net (Fig. 11.)

We will not detail the analysis of the Feedforward net, but it is important to check that the properties true in the application model are still holding in the Feedforward net.

If we take as example the property of the mutual exclusion of login and logout transitions, we can easily see in **Table 7.** and **Table 8.** that the places and the transitions belong are also listed in siphons, traps, P-invariants and T-invariants.

Table 7. Siphons and Traps from the feedforward behavior of the application.

Siphons	Traps
MessageToBeSent	MessageToBeSent
LoggedIn, LoggedOut	LoggedIn, LoggedOut

Of course, the Feedforward net is more complex and should also exhibit specific properties related to its own semantics. A very simple but important one is that whenever

the user triggers a transition to peek into the future (name starting with f1) immediately after a transition to come back to present (name starting with f2) and a transition to go into the future (name starting with f3) will be available. The analysis results in **Table 8.** demonstrate that a Feedforward net always verifies this fundamental property (any of such transitions is always in a T-Invariant with each other). If the results were different, it would have meant that developments faults have been added by the engineer when building the Feeforward net. In turn, the model would have to be modified so that the desired properties are present.

Table 8. Transitions and Place Invariants from the feedforward behavior of the application.

T-Invariants	**P-Invariants**
1 f4editMessage	1 LoggedIn, 1 LoggedOut
1 f1logout, 1 f3logout, 1 login	1 MessageToBeSent
1 f1login, 1 f2login	
1 editMessage	
1 f1editMessage, 1 f2editMessage	
1 f1sendAndClear, 1 f3sendAndClear	
1 f1sendAndClear, 1 f2sendAndClear	
1 f1logout, 1 f2logout	
1 login, 1 logout	
1 f1login, 1 f3login, 1 logout	
1 f1login, 1 f1logout, 1 f3login, 1 f3logout	
1 sendAndClear	
1 f1editMessage, 1 f3editMessage	
1 f1login, 1 f1logout, 1 f2login, 1 f3logout, 1 login	
1 f1login, 1 f2login, 1 login, 1 logout	

5.3 Formal Analysis of Fortunettes net and Interactive Application net Cooperation

Previous sections have demonstrated how formal analysis can be performed on for formal model of the Interactive Application and the formal model of Fortunettes independently. However, these two models communicate with eachother as explained in **Fig.** ***13***. Some adverse conditions must be avoided:

- The evolution of Fortunettes net does not alter the behavior of the interactive application (e.g. consuming resources (tokens) that would lead to starvation and thus deadlock of transitions)
- The evolution of the interactive application does not alter the behavior of Fortunettes rendering (e.g. as above consuming resources that would made this rendering unavailable after some interactions have occurred (wearing out)).

While identifying design solutions for the formal engineering of Fotunettes. Some options considered were to include Fortunettes pattern presented at the beginning of section 3.2 and instantiated for Login in **Fig. 8**, inside the Interactive Application net. While this would have simplified the overall structure of an application featuring Fortunettes feedforward, it would have:

- Made more complex the behavior of the entire application as fortunettes behavior would be merged with the one of the application;
  Made modification of the application behavior more cumbersome as only some part of the merged net describe the application;
- Might have introduced side effects on the behavior of the application (due to the evolutions of the Fortunettes net;
- Required deep formal analysis to demonstrate that the Fortunettes patterns added do not modify the initial behavior of the application and that the behavior of the application does not alter the expected Fortunettes behavior.

To avoid all these issues, we decided for two, independent models. **Fig. 8** explains how this independence (at model level) is bypassed at runtime by distributing the user events produced on the user interface to each model:

- User events performed directly on the user application widgets are distributed to both application and Fortunettes nets so they evolve concurrently and remain in synchronized (plain lines in **Fig. 8**);
- Fortunettes events are only distributed to the Fortunettes nets thus not impacting the behavior of the interactive application (dotted lines in **Fig. 8**).

6 Related work

6.1 User Experience and Feedforward

When users have a limited understanding of an application interface, they do not know in which cases to trust the application (Antifakos et al. 2005) (Muir 1994). To improve the user understanding, applications should provide in-situ explanations such as *what has happened* (feedback) and *what will happen* (feedforward) (Assad et al. 2007) , (Bellori & Edwards 2001) and (Lim & Dey 2010). Even though feedforward can present similar information as feedback, support for feedforward is less common. Nevertheless, feedforward has high potential to enhance the user experience since it can improve trust and prevent mistakes before the user action becomes final (Djajadiningrat et al. 2002) and (Schwarz et al. 2011).

Feedforward is often limited to basic labels or icons that only provide little amounts of information. More informative feedforward can be presented dynamically, based on the user's needs (Lafreniere et al. 2015). For example, ToolClips (Grossman & Fitzmaurice 2010), Stencils (Kelleher & Pausch (2005), and Side Views (Terry & Mynatt 2002) provide advanced tooltips based on hover events. In the context of gestural interfaces, OctoPocus (Bau & Mackay 2008) and Gestu-Wan (Rovelo et al 2015) dynamically visualize the next gestures available to the user to accommodate in-situ learning,

while TouchGhosts (Vanacken et al.) visualize the effects of user actions on the current system state.

6.2 Models for Generating Feedforward

In order to provide feedforward in a systematic way, a formal approach is needed. As highlighted in (Oliveira et al. 2017), many formal approaches to support the design, specification and verification of interactive systems have been proposed. That book chapter highlights four criteria to compare those approaches: 1) Modeling coverage (how much of the interactive systems can the notation describe); 2) Properties (and their type) supported; 3) Application of the methods in which domain; 4) Scalability (is the notation able to deal with large scale interactive systems).

In the past, a large number of models and notations have been contributed to support such formal approaches, including task models (Luyten et al. 2003) and (Martinie et al. 2015), finite state machines (Wood 1970), and Petri nets (Palanque et al. 1993). With respect to the modelling need of Fortunettes, the expressive power of the notation to be used heavily depends on the interactive application itself and does not require specific modelling power. In that regard, if the interactive application does not feature concurrent behavior, dynamic instantiation of objects and does not exhibit quantitative time behavior, automata would be adequate for describing Fortunettes behavior as demonstrated in (Coppers et al. 2019). If more complex behaviors need to be represented, more expression power will be required. The table 1 from the book chapter (Oliveira et al. 2017) would be then of great help to select the most adequate modeling notation depending on the features of the application and its interactions.

As Fortunettes feedforward concept is meant to be applied in a systematic way to all the interactions in an interactive system, Feedforward Nets need to cover all the components of the interactive systems (from the low-level interaction technique to the functional core) presented in the MIODMIT architecture (Cronel et al. 2018).

7 Limitations

This paper has presented one mean of engineering feedforward based on Fortunettes graphical and interaction design. The proposed solution involves the production of two ICO models for a single application. This increases significantly the modeling work even though part of the Feedforward Petri net is produced from the standard ICO model using the pattern presented above.

The use of a formal model based on Petri nets aims at detecting and recovering from development faults that might have been added to the standard application while adding the usability function feedforward. While the analysis section presented above demonstrates how this detection can be performed, it is important to note that the analysis only takes into account the underlying Petri net (as stated at the beginning of section 5). Going beyond that would require developing dedicated analysis methods for ICO models and more generally for high-level Petri nets. While some results have already been

published (Evangelista 2005), more work needs to be done and these contributions must be added to the analysis module of PetShop.
The proposed approach requires the use of PetShop at runtime. This might be seen as a limitation especially when the application has to be deployed and does not require modification of its appearance and behavior anymore. An alternative approach would be to translate the ICO models into event-based code, that would be compiled and deployed. Such a compilation approach is similar to what has been presented with automata instead of Petri nets (Coppers et al. 2019) or following the generic approach Petri nets compiling to event code (Palanque et al. 1993).
The current visual design of Fortunettes condenses feedforward information in a feedforward layer stacked behind each widget that will change. In order to prevent visual clutter in more complex applications with many interdependent widgets, we recommend designing more elaborate visualizations that can hierarchically aggregate feedforward information. To further reduce the burden on the user, feedforward could be limited to actions that the user is unfamiliar with, or only be shown when high confidence is desirable, by holding 'CTRL' on the keyboard for example.

8 Conclusion and perspectives

While research in the field of HCI focuses on adding more functionalities to the user interface, the interaction techniques and the interactive applications to improve usability and user experience, very little effort is devoted to transferring these improved interactions to the developers of interactive systems. For instance, papers proposing bubble cursor for improving target acquisition (Grossman & Balakrishnan 2005) or marking menus (Kurtenbach & Buxton 1994) to improve command selection do not present means for engineering these interaction techniques in a reliable and systematic way. We would argue that this is the reason for the limited take up of such HCI important contributions into real applications.

This paper has proposed an engineering method based on a formal description technique to support the systematic integration of Fortunettes concepts to provide interactive application with feedforward mechanisms. While the graphical and interaction design of Fortunettes might be improved and could be subject of future research, we have demonstrated that the use of a Petri nets-based approach limits the complexity of adding Fortunettes behavior to an existing application. We have also demonstrated that a formal approach can provide benefits in ensuring that the application with the additional feedforward behavior remains behaviorally compatible with the initial application.

The work presented in the present paper leads to extensions that will be addressed in future work. First, the current design of Fortunettes only deals with WIMP interaction techniques based on a set of identified widgets. While this can be seen as a strong limitation for current user interfaces targeting at better user experience, it is important to note that many applications are still widget-based. In some critical domains it is even not possible to embed other types of interfaces as required by the ARINC 661 specification standard (Arinc 2013) for user interfaces of cockpits of large civil aircrafts. We have previously worked on the formal description of User Application, user interface

widgets and user interfaces servers using Petri net based description (Barboni et al. 2006) and that early work can directly benefit from the work presented in the paper. This means that adding the feedforward usability function to those user applications will result in very limited work (as the Feedforward net is built upon the original behavior and is described with the same language) and would come with assurance means to guarantee their correct behavior. Beyond, we can also exploit the Petri net models to assess the usability of Fortunettes interaction technique as introduced in (Palanque et al. 2011) for other interaction techniques.

Second, the current behavior of Fortunettes is to offer the possibility to the user to look only one step into the future. The model-based behavior presented in the paper could be exploited further to look into several step or even to look at the eventual end of the execution, as introduced in (Palanque et al. 1995). For instance it would be possible to identify a widget (via formal analysis) that would become unavailable forever in five steps from the current state of the application .While graphical design and interaction will be clearly a difficult challenge, the engineering of such applications could be reachable via the analysis of the formal models.

Acknowledgments

This work was funded by the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” and the Research Foundation - Flanders (FWO, project G0E7317N End-User Development of Intelligible Internet-of-Things Ob- jects and Applications).

References

(Antifakos et al. 2005) Antifakos, S., Kern, N., Schiele, B., & Schwaninger, A. (2005). *Towards improving trust in context-aware systems by displaying system confidence*. 9. https://doi.org/10.1145/1085777.1085780

(Arinc 2013) ARINC 661. Cockpit Display System Interfaces to User Systems. ARINC Specification 661-5. AEEC, 2013

(Assad et al. 2007) Assad, M., Carmichael, D. J., Kay, J., & Kummerfeld, B. (2007). PersonisAD: Distributed, Active, Scrutable Model Framework for Context-Aware Services. In A. LaMarca, M. Langheinrich, & K. N. Truong (Eds.), *Pervasive Computing* (Vol. 4480, pp. 55–72). Retrieved from http://link.springer.com/10.1007/978-3-540-72037-9_4

(Avizienis et al. 2004) Avižienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and tax-onomy of dependable and secure computing. Dependable and Secure Computing, IEEE Transactions on, 1(1), 11-33

(Badouel et al. 1995) Badouel E., Bernardinello L., Darondeau P. (1995) Polynomial algorithms for the synthesis of bounded nets. In: Mosses P.D., Nielsen M., Schwartzbach M.I. (eds) TAPSOFT '95: Theory and Practice of Software Development. CAAP 1995. Lecture Notes in Computer Science, vol 915. Springer, Berlin, Heidelberg

(Barboni et al. 2006) Barboni E., Conversy S., Navarre D. & Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification.

13th conf. on Design Spec. and Verif. of Interactive Systems (DSVIS 2006), LNCS Springer Verlag. p25-38

(Bass et al. 2020) Bass L. Clements P. and Kazman R. Software architecture in practice. Addison-Wesley, 2020.

(Bastide et al. 1993) Bastide R., Sibertin-Blanc C., Palanque P. Cooperative objects: A concurrent, petri-net based, object-oriented language. IEEE Systems Man and Cybernetics Conference-SMC 1993, 286-291

(Bastide & Palanque 1999) Bastide *R. and* Palanque P.*:* A Visual and Formal Glue between Application and Interaction. J. Vis. Lang. Comput. 10(4)*: 481-507 (1999)*

(Bastide et al. 2002) Bastide, R., Navarre, D., Palanque, P.: A Model-based Tool for Interactive Prototyping of Highly Interactive Applications. CHI ’02 Extended Abstracts on Human Factors in Computing Systems. pp. 516–517. ACM, USA (2002).

(Bastide et al. 2004) Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004, USA, ACM Press.

(Bau & Mackay 2008) Bau, O., & Mackay, W. E. (2008). *OctoPocus: A dynamic guide for learning gesture-based command sets*. 37. https://doi.org/10.1145/1449715.1449724

(Bellori & Edwards 2001) Bellotti, V., & Edwards, K. (2001). Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction*, *16*(2), 193–212. https://doi.org/10.1207/S15327051HCI16234_05

(Berlage 1994) Berlage T. 1994. A selective undo mechanism for graphical user interfaces based on command objects. ACM Trans. Comput.-Hum. Interact. 1, 3 (September 1994), 269–294. DOI:https://doi.org/10.1145/196699.196721

(Buschmann 1996) Buschmann, F. (1996) *Pattern-Oriented Software Architecture*. Wiley.

(Canfora & Cerulo 2005) Canfora G. and Cerulo L. 2005. How Crosscutting Concerns Evolve in JHotDraw. In Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP '05). IEEE Computer Society, Washington, DC, USA, 65-73.

(Coppers et al. 2019) Coppers, S., Luyten, K., Vanacken, D., Navarre, D., Palanque, P., Gris, C. Fortunettes: Feedforward about the Future State of GUI Widgets. Proceedings of the ACM on Human-Computer Interaction EICS, vol:3. 2019, ACM SIGCHI.

(Cronel et al. 2018) Cronel M., Dumas B., Palanque P., Canny A. 2018. MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In Proc. of IFIP TC13.2 Conference on Human Centered Software Engineering, HCSE 2018, 109--129.

(David & Alla 1992) David R., Alla H. Petri nets and grafcet - tools for modelling discrete event systems. Prentice Hall 1992, ISBN 978-0-13-327537-7, pp. I-XII, 1-339

(Dietze et al. 2007) Dietze R., Kudlek M., Kummer O. Decidability Problems of a Basic Class of Object Nets. Fundam. Inform. 79(3-4): 295-302 (2007)

(Djajadiningrat et al. 2002) Djajadiningrat T., Kees Overbeeke, and Stephan Wensveen. 2002. But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback. Conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '02). ACM, New York, NY, USA, 285-291.

(Evangelista 2005) Evangelista S. High Level Petri Nets Analysis with Helena. In: Ciardo G., Darondeau P. (eds) Applications and Theory of Petri Nets 2005. ICATPN 2005. Lecture Notes in Computer Science, vol 3536. Springer, Berlin, Heidelberg

(Fayollas 2017) Fayollas C., Palanque P., Fabre J-C., Martinie C., Déléris Y. (2017) Dealing with Faults During Operations: Beyond Classical Use of Formal Methods. The Handbook of Formal Methods in Human-Computer Interaction. Human–Computer Interaction Series. Springer.

(Grossman & Balakrishnan 2005) Grossman T. and Balakrishnan R. 2005. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05). ACM, DL, 281-290.

(Grossman & Fitzmaurice 2010) Grossman, T., & Fitzmaurice, G. (2010). *ToolClips: An investigation of contextual video assistance for functionality understanding*. 1515. https://doi.org/10.1145/1753326.1753552

(Hamon et al. 2013) Hamon A., Palanque P., Silva J-L., Deleris Y., and Barboni E. 2013. Formal description of multi-touch interactions. 5th ACM SIGCHI symposium on Engineering Interactive Computing Systems (EICS '13). ACM, 207–216.

(Kelleher & Pausch 2005) Kelleher, C., & Pausch, R. (2005). *Stencils-based tutorials: Design and evaluation*. 541. https://doi.org/10.1145/1054972.1055047

(Kurtenbach & Buxton 1994) Kurtenbach G. and Buxton W. 1994. User learning and performance with marking menus. Conference on Human Factors in Computing Systems (CHI '94). ACM DL, 258-264.

(Lafreniere et al. 2015) Lafreniere, B., Chilana, P. K., Fourney, A., & Terry, M. A. (2015). These Aren't the Commands You're Looking For*: Addressing False Feedforward in Feature-Rich Software*. 619–628. https://doi.org/10.1145/2807442.2807482

(Lee & Yamada 2010) Lee S. & Yamada Y. (2010) Strategy on Safety Function Implementation: Case Study Involving Risk Assessment and Functional Safety Analysis for a Power Assist System, Advanced Robotics, 24:13, 1791-1811

(Lim & Dey 2010) Lim, B. Y., & Dey, A. K. (2010). *Toolkit to support intelligibility in context-aware applications*. 13. https://doi.org/10.1145/1864349.1864353

(Luyten et al. 2003) Luyten, K., Clerckx, T., Coninx, K., & Vanderdonckt, J. (2003). Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In J. A. Jorge, N. Jardim Nunes, & J. Falcão e Cunha (Eds.), *Interactive Systems. Design, Specification, and Verification* (pp. 203–217). Berlin, Heidelberg: Springer Berlin Heidelberg.

(Martinie et al. 2015) Martinie, C., Navarre, D., Palanque, P., & Fayollas, C. (2015). *A generic tool-supported framework for coupling task models and interactive applications*. 244–253. https://doi.org/10.1145/2774225.2774845

(Muir 1994) Muir, B. M. (1994). Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics*, *37*(11), 1905–1922. https://doi.org/10.1080/00140139408964957

(Murata 1989) Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE (Volume: 77 , Issue: 4 , Apr 1989)

(Navarre et al. 2009) 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 Trans. Comput.-Hum. Interact., 16(4), 18:1–18:56. 2009.

(Norman 1988) Norman, D. A. The Psychology Of Everyday Things. Basic Books, New York, USA, June 1988

(Norman 2003) Norman, D. A. The design of everyday things: Revised and expanded edition, Basic Books, New York, USA, 2013

(Oliveira et al. 2017) Oliveira Prates R., Palanque P., Weyers B., Bowen J., Dix A. State of the Art on Formal Methods for Interactive Systems. In (Weyers et al. 2017) pp.: 3-55

(Palanque & Bastide 1993) Palanque P., Bastide R., Dourte L.Contextual Help for Free with Formal Dialogue Design. In Proceedings of HCI International (2) 1993: 615-620

(Palanque et al. 1993) Palanque P.A., Bastide R., Dourte L., Sibertin-Blanc C. (1993) Design of user-driven interfaces using Petri nets and objects. In: Rolland C., Bodart F., Cauvet C. (eds) Advanced Information Systems Engineering. CAiSE 1993. Lecture Notes in Computer Science, vol 685. Springer, Berlin, Heidelberg

(Palanque et al. 1995) Palanque P., Bastide R., Sengès V. Validating interactive system design through the verification of formal task and system models. IFIP WG 2.7, working conference Engineering HCI, 1995, Springer,189-212

(Palanque et al. 2006) Palanque P., Bernhaupt R., Navarre D., Ould M. & Winckler M. Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. Ninth Int. Conference on Space Operations, Italy, June 18-22, 2006.

(Palanque et al. 2009) Palanque P., Ladry J-F, Navarre D. & Barboni E. High-Fidelity Prototyping of Interactive Systems can be Formal too 13th Int. Conf. on Human-Computer Interaction (HCI International 2009) LNCS, Springer.

(Palanque et al. 2011) Palanque P., Barboni E., Martinie C., Navarre D., and Winckler M. 2011. A model-based approach for supporting engineering usability evaluation of interaction techniques. 3rd ACM SIGCHI symposium on Engineering interactive computing systems (EICS '11). ACM, 21–30.

(Polet et al. 2002) Polet, P, Vanderhaegen, F, and Wieringa, P. Theory of safety related violation of system barriers. Cognition Technology & Work, 4, 3, 171-179. 2002

(Reason 1990) Reason, J. (1990). Human Error, Cambridge University Press

(Rovelo et al 2015) Rovelo, G., Degraen, D., Vanacken, D., Luyten, K., & Coninx, K. (2015). Gestu-Wan—An Intelligible Mid-Air Gesture Guidance System for Walk-up-and-Use Displays. In J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, & M. Winckler (Eds.), *Human-Computer Interaction – INTERACT 2015* (Vol. 9297, pp. 368–386). https://doi.org/10.1007/978-3-319-22668-2_28

(Sadasivan et al. 2005) Sadasivan S., Joel S. Greenstein, Anand K. Gramopadhye, and Andrew T. Duchowski. 2005. Use of eye movements as feedforward training for a synthetic aircraft inspection task. Conference on Human Factors in Computing Systems (CHI '05). ACM, 141-149.

(Schwarz et al. 2011) Schwarz, J., Mankoff, J., & Hudson, S. (2011). Monte Carlo Methods for Managing Interactive State, Action and Feedback Under Uncertainty. *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 235–244. https://doi.org/10.1145/2047196.2047227

(Silva et al. 2013) Silva J-L, Fayollas C., Hamon A., Palanque P., Martinie C., Barboni E. Analysis of WIMP and Post WIMP Interactive Systems based on Formal Specification. ECEASST 69 (2013)

(Terry & Mynatt 2002) Terry, M., & Mynatt, E. D. (2002). *Side views: Persistent, on-demand previews for open-ended tasks*. 71. https://doi.org/10.1145/571985.571996

(Vanacken et al.) Vanacken, D., Luyten, K., & Coninx, K. (n.d.). TouchGhosts: Guides for Improving Visibility of Multi-Touch Interaction.

(Vermeulen et al. 2013) Vermeulen J., Luyten K., Elise van den Hoven, and Karin Coninx. 2013. Crossing the bridge over Norman's Gulf of Execution: revealing feedforward's true identity. SIGCHI Conference on Human Factors in Computing Systems (CHI '13). ACM, USA, 1931-1940

Weyers B., Bowen J., Dix A., Palanque P. (2017) The Handbook of Formal Methods in Human-Computer Interaction. Human–Computer Interaction Series. Springer, Cham

(Wood 1970) Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, *13*(10), 591–606. https://doi.org/10.1145/355598.362773

(Yoon et al. 2015) Yoon C., Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. 2015. Enabling security functions with SDN. Comput. Netw. 85, C (July 2015), 19-35.

(Zhang & Wang 2000) Zhang M. and Wang K. 2000. Implementing Undo/Redo in PDF Studio Using Object-Oriented Design Pattern. In Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia'00) (TOOLS '00). IEEE Computer Society, USA, 58.