Fortune Nets for Fortunettes: Formal, Petri Nets-Based, Engineering of Feedforward for GUI Widgets

Peer-reviewed author version

Navarre, David; Palanque, Philippe; COPPERS, Sven; LUYTEN, Kris & VANACKEN, Davy (2020) Fortune Nets for Fortunettes: Formal, Petri Nets-Based, Engineering of Feedforward for GUI Widgets. In: Sekerinski, Emil; Moreira Nelma; Oliveira, Jose N. et al (Ed.). Formal Methods FM 2019, Springer, Cham, p. 503 -519.

DOI: 10.1007/978-3-030-54994-7_36
Handle: http://hdl.handle.net/1942/31947

Fortune Nets for Fortunettes: Formal, Petri nets-based, Engineering of Feedforward for GUI Widgets

David Navarre[1[0000-0002-2900-2056]], Philippe Palanque[1-2[0000-0002-5381-971X]], Sven Coppers[3[0000-0002-5734-8898]], Kris Luyten[3[0000-0002-4194-1101]] and Davy Vanacken[3[0000-0001-8436-5119]]

[1] ICS-IRIT, University of Toulouse, 118 Route de Narbonne, F-31062, Toulouse, France
[2] Technical University Eindhoven, Department of Industrial Design, Eindhoven, Netherlands
[3] Hasselt University - tUL - Flanders Make, Expertise Centre for Digital Media, Diepenbeek Belgium
{navarre, palanque}@irit.fr ; firstname.lastname@uhasselt.be

Abstract. Feedback and feedforward are two fundamental mechanisms that supports 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. 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. This paper proposes the use of an existing formal notation for describing the behavior of interactive applications and how to exploit that formal model to extend the behavior to offer feedforward. We use a small login example to demonstrate the process and the results.

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

1 Introduction

Feedback and feedforward are two fundamental mechanisms supporting 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. Automatic feedforward presents in a systematic way to the users what can be done without requiring any dedicated action (e.g. greying out an interactive object that is not available). Automatic feedforward is often available in well-designed interfaces. User-triggered feedforward provides localized, contextual information to the users related to the actions that they envision triggering (e.g. painting temporarily a selected object in yellow while hovering over the yellow button for painting objects). User-triggered feedforward is usually not available in user interface, 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 [25], the authors exploit Norman's activity theory [17] to explain the importance and the impact of providing users with feedforward in user interfaces, especially in the action selection phase. In poorly designed systems, that kind of user activity can be very cumbersome especially in the upper part of the model of the activity theory (also called semantic distance).

Fig. 1 presents a typical system offering limited feedforward. In that system (Microsoft Word) some of the commands for changing text graphical attributes do not propose feedforward (see **Fig. 1** b) while others do (see **Fig. 1** c)). **Fig. 1** a) presents a snapshot of MS Word software with the word Fortunettes selected and highlighted. In that version of MS Word, when some text is selected, a contextual pop-up menu appears next to the selected text. In **Fig. 1** a) the cursor has been moved far away from the selected text and thus no pop-up menu is visible. In **Fig. 1** b) the pop-up menu is displayed and the mouse cursor hovers over the Bold command to change the presentation of the text to Bold. However, in that case, no feedforward is presented so it is not possible to see how the text will be if the Bold command is performed. Surprisingly, **Fig. 1** c) highlights the fact that for altering the color of the selecting text, hovering over one of the colors displayed in the pop-up menu applies directly the hovered over color to the selected text, thus providing users with feedforward on the color attribute of the text.

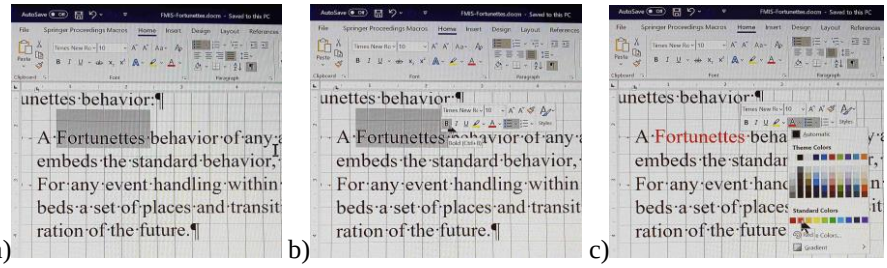

Fig. 1. Inconsistent availability of feedforward in Microsoft Word (Office 2016)

One of the questions that arises immediately is: why such a sophisticated tool as MS Word is not offering feedforward mechanisms for all the functions or at least to all the similar functions (e.g. changing attributes of selected text).

While, as highlighted in [10] and [25], the design of feedforward is an issue. We would argue that its specification and its implementation are the key problem to solve when it is considered as a potential function to add to the system. In that case, we would argue that feedforward is a **usability function** using the pending concept of **security function** [26] or **safety function** [13]. While a safety function can be defined as a function added to a system to prevent undesired safety problems, we would define a usability function as a function added to an interactive system to prevent undesired usability problems. Within this context, feedforward can be considered as a function similar to "undo" and thus requires complex implementation due to its crosscutting nature [13].

This paper argues for the use of a formal approach for the specification and the implementation of feedforward in a systematic way. We present how the expressive power of high-level Petri nets such as ICOs [16] can describe feedforward and how the resulting models are amenable to verification (to identify and check properties on the system offering feedforward). In a nutshell we propose to produce a Petri net model (called *Fortune Net*) in addition to the model describing the behavior of the application. We also argue that a formal model of the initial application can be extended in a systematic way to include feedforward functionality, thus reducing development cost of such a usability function.

This paper is an extension of the work done in [7] to offer feedforward mechanisms in a more general context. 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 Fortune Nets ones. Section 6 concludes the paper and highlight paths for future work.

2 Fortunettes: Design, Foundations and Use

The origin of Fortunettes [7] is the need of providing feedforward about the future state of an application. When including a feedforward usability function in the GUI, the feedforward information does not need to be presented permanently (to avoid visual overload and cluttering of the UI) but instead we propose this specific information display to be triggered by the user on demand (when needed). In our approach, exploring the future may be seen as a four steps process:

- **Look at the present,** when the user explores visually the user interface elements;
- **Peek into the future**, when the user is considering performing an action;
- **Go to the future**, when the user confirms and actually executes the considered action;
- **Return to the present**, when the user is no longer considering the execution of that action.

The choice has been made of providing such feedforward at widget-level as it makes it easier to reuse for any widget-based application. **Fig. 2** shows an example of this kind of widget-level feedforward: when the user hovers over the `Login` button (that is currently enabled), the button `Logout` and the text box (that are currently disabled), show their future state in terms of availability (the button `Logout` and the textbox will become enabled if the user clicks the button `Login`, while the button `Login` will become disabled). With this information, the user knows that to enable the `Logout` button, the `Login` button me be pressed first.

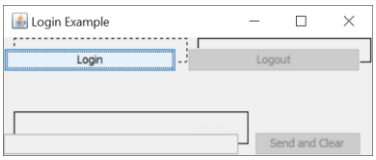

Fig. 2. Illustration of the Fortunettes concept using the case study.

The main idea of Fortunettes is to provide the user with an answer to "What will happen if I do that?", by presenting what the result of the user action will be, before the action is actually performed. It thus requires the widgets to be able to present their future state in addition to their current state (enabled or disabled).

As presented in [7], the user interface of the application presented in **Fig. 2** is the following:

- The application is composed of four widgets (the three buttons and a textbox),
- The current state of the widgets is displayed on the forefront, the login button is enabled, "Logout" and "Send and Clear" ones are disabled, and the textbox is disabled too.
- In order to present the feedforward information, users have to hover over the widget of interest. In **Fig. 2**, the "login" button is hovered and the background display of each widget presents the feedforward information showing the state of the application if the user clicks on the login button. Current feedforward display tells the user that "login" button will be disabled, the textbox will become enabled, "logout" will become enabled and "Send and Clear" will remain disabled. Indeed, as the status of "Send and Clear" will remain the same, no additional feedforward display is presented. We follow here the parsimony principle of user interface designs.

The design choice presented here is one example of the many possible designs of Fortunettes: every widget is decorated with borders to express its future availability (full lines for enabled, dashed lines for disabled) and/or its future values.

This design will not be further discussed as the focus of this paper is on formal description and engineering support. These two aspects are particularly important as the introduction of Fortunettes increases the complexity of the development of an application, and, by consequence its reliability.

3 Illustrative example

We illustrate the use of the Fortunettes approach with a simple application (as illustrated by **Fig. 3**) that behaves as follows: when the user is logged in, a message can be written in the textbox or the user can log out. To ensure that the message only contains letters, the edited text is filtered, removing any other characters (numbers, special characters…). If the textbox is not empty, the message can be sent. Sending the message or logging out clear the textbox.

Fig. 3. Screen shots of the illustrative application. On the left, the user is logged out, on the right, the user is logged in and a message is being edited and ready to be sent.

4 Modelling of Fortunettes behavior

To support the engineering of interactive applications offering a feedforward usability function based on Fortunettes, we propose an approach based on a formal description technique called Interactive Cooperative Objects (ICO). We firstly present in this Section the formal description technique, then we present how it is possible to derive the feedforward behavior of the application from the existing model of the application behavior.

4.1 ICO formal description technique

The ICO formalism is a formal description technique dedicated to the modeling and the implementation of event-driven interfaces [16], using 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 called Cooperative Objects (CO) [4]. In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behavior of the object, a presentation part (i.e. the graphical interface), and two functions (the activation function and the rendering function) which connects the cooperative object and the presentation part.

An ICO specification fully describes the potential interactions that users may have with the 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 [16], space command and control ground systems [20], or interactive military [3] or civil cockpits [2].

The ICO notation is fully supported by a CASE tool called PetShop [5, 21]. All the models presented in the two next Sections (4 and 5) have been edited, simulated and analyzed using PetShop tool.

4.2 Principle of Fortunettes feedforward modelling using ICO

As stated in Section 2, engineering an application with feedforward capabilities requires to handle extra interaction events (at least three, depending on the widget type). These events allow the user to **peek into the future**, to **go to the future** or to **return to the present**, without affecting the standard behavior of the application, as the objective is

to enhance the application (with feedforward) and not to change it. This design choice impacts the modelling of feedforward behavior:

- The feedforward behavior of any application is modelled as an independent object that embeds the standard behavior (as a copy), making it fully compatible with the original application behavior. This Petri net model is called the **Fortune Net** as it allows users to look into the future of the application.
- For any event handling within the standard behavior, the feedforward behavior embeds a pattern described in Petri nets (a set of places and connected transitions) that models the exploration of the future states. The important aspect in this modelling principle is that we exploit the behavior of the application to forecast the future states of the application if the user decides to use feedforward function.

To illustrate these two points, we use an excerpt of the complete behavior presented in the next Section (4.3) that only concerns the `login` action on the user interface (as shown by **Fig. 4**).

Fig. 4. Excerpt from the Petri net model of the standard behavior of the application: event handling of the login action. In the transition, the text on the left describes the name of the transition while the text on the right describes the name of the event associated the transition.

In Fig. 4, 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 (code not represented here).

When introducing the Fortunettes view on this action, the three base actions defined in Section 2 (peek into the future, go to the future and return to the present) are represented as three extra event handlers, as shown on Fig. 5, where event handlers {`FloginPerformed`, `UFloginPerformed`, `InFloginPerformed`} are generated from the event handler `loginPerformed`. In this paper, the name of the generated event handlers for handling Fortunettes mode are built with 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).

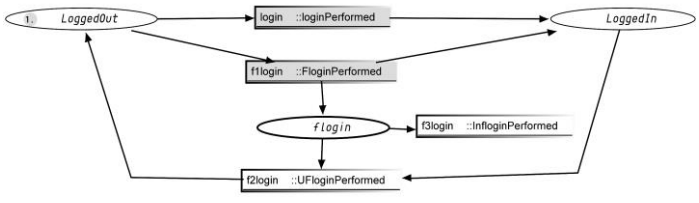

Fig. 5. Extracted from the feedforward behavior of the application: event handling of the login action and peek into its future.

On Fig. 5, 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:

- The user decides to really perform the `login` action (using the `login button`), producing 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 to not perform the login action producing 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 a feedforward behavior for events that do not handle values or when the widgets are simple such as button. For more complex events, or when the underlying widgets are more complicated, this pattern has to be modified/extended:

- When values are handled by the action of the widget, it is not always possible to peek into the future of these values. One possible improvement is to proceed in two steps. When entering the feedforward mode, an envisioned value must be produced (decided at design time for instance) and when the user really performs the action, a substitution must be done between the envisioned value and the real value. In the feedforward behavior, this can be done by moving tokens (if it was the case in the login example, the first token put in place `LoggedIn` by transition `f1login` would have a design time envisioned value, and when `f3login` would be fired, this token would have been removed and replaced by one holding the correct value).
- When the widget is more complex (in our case, the complexity is related to the event production), extra event handlers may be introduced. For instance, when using a classical textbox, one may be interested by the end of the text edition (validation) and not by the whole process of typing in the text. In this case, in the standard behavior of the application, the only handled event would be 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 to allow the rendering of text filtering.

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.

4.3 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. 6 presents the entire behavior of the illustrative example. It may be divided into two parts: the upper part is dedicated to login actions and the lower part is dedicated to the message handling.

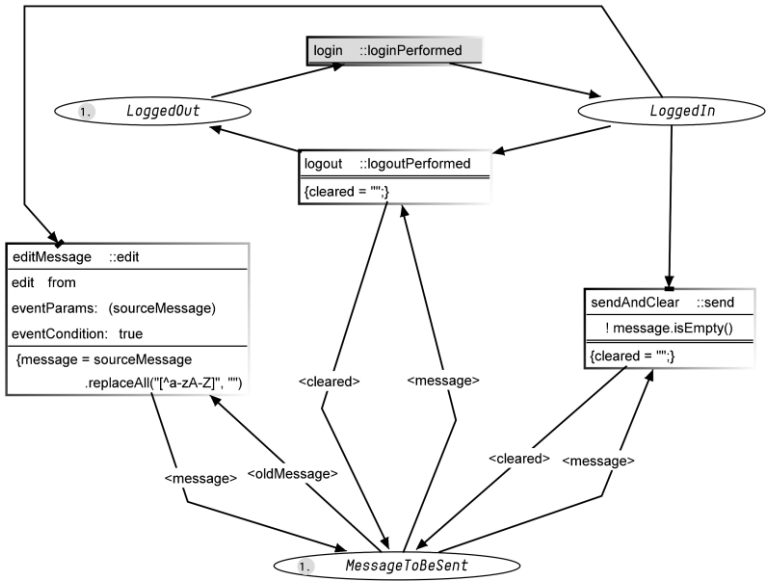

Fig. 6. Behavior of the Login example using the ICO formal description technique.

The upper part of **Fig. 6** models what has been explained in the beginning of the Section (see Fig. 4) 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. 6** 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. 7 illustrates how feedforward information can be displayed using Fortunettes. **Fig. 8**, **Table 3** and **Table 4** fully describe the feedforward part of the application. The behavior presented by **Fig. 8** is structured similarly to the standard behavior, the upper part being dedicated to the login actions and the lower part, to the message editing.

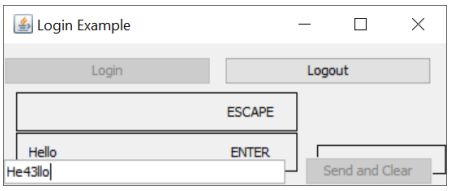

Fig. 7. Illustration of the text filtering while typing in feedforward mode

This Fortune 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. 8**). 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. 7**.

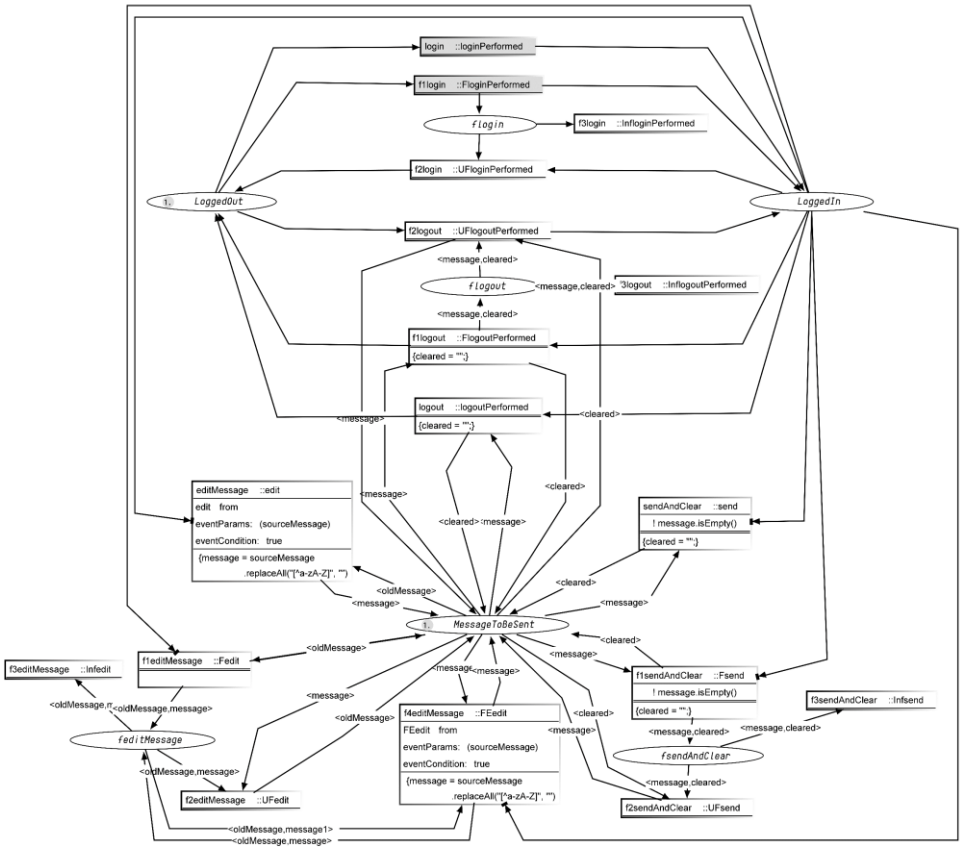

Fig. 8. The Fortune Net describing the feedforward behavior of the Login example using the ICO formal description technique.

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. 7**, 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

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. 7)

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 interesting joint behavior between the standard behavior of the application and its Fortunettes ones is highlighted on **Fig. 7**. 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.

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 Fortune Net) has multiple implications. First, the standard application models must exhibit some properties and it is important to check that they are true. Second, the Fortune 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 Fortune 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 available in the Fortune Net. This is only an example of the generic properties that have to be checked when a feedforward usability function is added to an application.
With ICOs, as detailed in [24] and [19], there are two different 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 [15].
- The analysis of the high-level Petri net (ICO) but this requires manual demonstrations as some of the properties are undecidable [9].

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 with objects 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 [4].

5.1 Formal analysis of the model of the standard behavior (Fig. 6)

Table 5 presents the list of traps and siphons of the model in **Fig. 6**[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 [8]. The fact that all the places 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. 6**).

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	

[1] The computing of the results in those tables was done using Petshop tool and are not presented due to space constraints. How to make such computing is presented in [8].

In terms of behavior, transitions login and transition 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. 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 Fortune Net (Fig. 8)

We will not detail the analysis of the Fortune Net, but it is important to check that the properties true in the application model are still holding in the Fortune 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 Fortune 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 Fortune Net always verifies this fundamental property (any of such transitions is always in a T-Invariant with each other).

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	

6 Related work

As highlighted in [22] 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).

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. With that respect, 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 [7]. If more complex behaviors need to be represented, more expression power will be required. The table 1 from the book chapter [22] would be then of great help to select the modeling notation.

As Fortunettes feedforward concept is meant to be applied in a systematic way to all the interactions in an interactive system, Fortune Nets need to cover all the aspects of the interactive (from the low-level interaction technique to the functional core according to the MIODMIT architecture [14]. We have only presented here Fortunettes at the application level, but all the layers of the architectures should be taken into account.

7 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 work is spent on transferring these improved interactions to the developers of interactive systems. For instance, papers proposing bubble cursor for improving target acquisition [11] or marking menus [12] to improve command selection do not present means for engineering these interaction techniques in a reliable and systematic way.

This paper has proposed an engineering method based on formal methods 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 should 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 [1] for user interfaces of cockpits of large civil aircrafts. We have previously worked on the formal description of User Application, user interface widgets and servers using Petri net based description [2] 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 Fortune 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.

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 [19]. 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.

References

1. ARINC 661. Cockpit Display System Interfaces to User Systems. ARINC Specification 661-5. AEEC, 2013
2. 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
3. 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.
4. 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
5. 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).
6. Canfora G. and Luigi Cerulo. 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.
7. 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 vol:3. ACM SIGCHI.
8. 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

9. Dietze R., Kudlek M., Kummer O. Decidability Problems of a Basic Class of Object Nets. Fundam. Inform. 79(3-4): 295-302 (2007)
10. 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.
11. 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.
12. 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.
13. 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
14. Martin Cronel, Bruno Dumas, Philippe A. Palanque, Alexandre Canny. 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.
15. Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE (Volume: 77 , Issue: 4 , Apr 1989)
16. 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.
17. Norman, D. A. The Psychology Of Everyday Things. Basic Books, New York, USA, June 1988
18. Palanque P., Bastide R., Dourte L.Contextual Help for Free with Formal Dialogue Design. In Proceedings of HCI International (2) 1993: 615-620
19. 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
20. 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.
21. 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.
22. Raquel Oliveira Prates, Philippe A. Palanque, Benjamin Weyers, Judy Bowen, Alan J. Dix. State of the Art on Formal Methods for Interactive Systems. Handbook of Formal Methods in Human-Computer Interaction 2017: 3-55
23. 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.
24. 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)
25. Vermeulen J., Kris Luyten, 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
26. 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.