Animated Transitions between User Interface Views

Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt
Université catholique de Louvain, Louvain School of Management
Louvain Interaction Laboratory, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{vivian.genaromotti, jean.vanderdonckt}@uclouvain.be – Phone: +32 10 478525

ABSTRACT
User interface development life cycle often involve several different views of the user interface over time either at the same level of abstraction or at different levels of abstraction. The relationship between these different views is often supported by tiling coordinated windows containing these related views simultaneously, thus leaving the developer with the responsibility to effectively and efficiently link the corresponding elements of these different views. This paper attempts to overcome the shortcomings posed by the coordinated visualization of multiple views by providing UsiView, a user interface rendering engine in which one single window ensures an animated transition between these different user interface views dynamically: an internal view, an external view, and a conceptual view. Examples include the following cases: an authoring environment ensures an animated transition between an internal view (e.g., HTML5) and its external view (e.g., a web page), an Integrated Development Environment ensures an animated transition between its conceptual view and its external view; a model-driven engineering environment ensures an animated transition between the conceptual view at different levels of abstraction, e.g., from task to abstract user interface to concrete user interface until final user interface. The paper discusses the potential advantages of using animated transitions between user interface views during the development life cycle.

Categories and Subject Descriptors
D2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [**Software Engineering**]: Miscellaneous – *Rapid Prototyping; reusable software*. H.5.1 [**Information interfaces and presentation**]: Multimedia Information Systems – *Animations*. H5.2 [**Information interfaces and presentation**]: User Interfaces – *User-centered design*.

General Terms
Design, Experimentation, Human Factors, Verification.

Keywords
Animated transition, animation, model evolution animation, user interface development method, user interface view.

1. INTRODUCTION
Authoring a web User Interface (UI) typically requires the developer to write the UI code in a markup (e.g., HTML5) and/or a programming language (e.g., JavaScript) and test its rendering until a satisfactory result is obtained. In model-based UI design, the process requires the designer to model the UI of concern and test the final UI resulting from it until a satisfactory result is obtained. Similarly, in model-driven UI engineering, the transformational approach requires the engineer to apply the right transformations from one level of abstraction to another [25], and test the final UI issued from this approach until a satisfactory result is obtained. In these three cases, analysts, designers and developers constantly oscillate across multiple *user interface views*, that represent a final UI according to a certain representation either at the same level of abstraction or at a superior level of abstraction [9].

For *beginners*, this back-and-forth process is inconvenient not only because they are not yet familiar with the different views but also because it is hard for them to relate the various elements contained in the different views. Moreover, modifying one element in a UI does not facilitate identifying the potential implications in the other views since no connection exists across views.

For *expert users*, this process is also inconvenient because it forces them to perpetually maintain a correspondence between the views. These views are potentially expressed with different notations, specification, formal methods or models and do not offer any immediate feedback regarding the views' transitions. Moreover when a complex UI syntax is used, it does not lend itself readily to proofreading the UI rendering when the UI syntax differs considerably from the rendered UI.

Consequently, UI engineering processes typically involve several different views of the UI of concern throughout the UI development life cycle. Such views are represented with heterogeneous semantics, syntaxes, and stylistics, which requires the establishment and maintenance of the correspondence between these views. To address the shortcomings induced by this back-and-forth process, we introduce *UsiView*, a UI rendering engine that ensures a smooth progression from one view to another using animated transitions.

The remainder of this paper is structured as follows: the Section 2 provides some theoretical and historical background on UI views. The methodology for defining the animated transitions between UI views are motivated and presented in Section 3. The software architecture of the rendering engine is explained in Section 4. Finally, Section 5 delivers a conclusion with the main points of this research and presents some future avenues.

2. BACKGROUND
2.1 Theoretical background: the UI view
We hereby define a *UI view* as any representation of a final UI involved in a UI development life cycle. A UI view may be textual, graphical or both, based on a data structure or not. By observing UI development methods and life cycles, UI views can be classified into three categories (Fig. 1):

1. *Conceptual View* (CV): describes a conceptual representation of a UI of interest based on semantics, syntax, and stylistics. Typical examples include: UI models for domain, functional core, resources, and dynamic aspects. A conceptual view is the

AVI '12, May 21-25, 2012, Capri Island, Italy Copyright © 2012 ACM 978-1-4503-1287-5/12/05... $10.00

designer's view at early stage.

2. *Internal View* (IV): consists of the UI code in any programming or markup language. An internal view is the typical developer view for developing a particular UI.
3. *External view* (EV): refers to the final UI that is visible and executable by the end user.

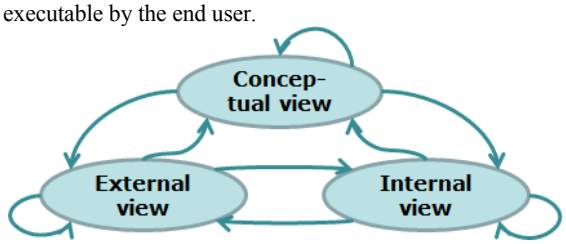

Figure 1. Possible paths between UI views.

During the development life cycle, also at run-time (e.g., in end-user programming), various UI stakeholders can create, retrieve, modify, delete, or simply execute any UI view or view element: for instance, while a designer is responsible for the conceptual view, the developer is responsible for the internal view, and the end user accesses the external view. Consequently, nine development paths are possible (Fig. 1):

1. *External view to Internal view*: consists in drawing to the largest extent possible the final UI, e.g., in a graphical editor of an Integrated Development Environment (IDE), and to derive code from this drawing. Presentation aspects are generally well covered since they are static, as opposed to dynamic aspects (e.g., navigation, behavior) that are difficult to draw, and thus require a conceptual view. Graphical UI (GUI) builders are better at coding presentation than dialogue.
2. *Internal view to External view*: consists in obtaining the UI rendering from its application/UI code, thus requiring compilation or interpretation.
3. *Conceptual view to External view*: exploits one or many UI models in order to derive the UI rendering as straightforwardly as possible. Model based design of UIs [24] adheres to this frequently used path.
4. *Conceptual view to Internal view*: exploits one or many UI models in order to derive the UI code as straightforwardly as possible. Model-based design is also relevant. Model-to-code (M2C) transformations are also considered in this category.
5. *External view to Conceptual view*: applies reverse engineering techniques in order to recover any UI model from its visual rendering. For instance, it is possible to regenerate a UI presentation model from a screen shot, an image, or a video.
6. *Internal view to Conceptual view*: applies reverse engineering techniques in order to recover any UI model from its code. For instance, a UI presentation model from its HTML code.
7. *Loop on Conceptual view*: applies model-based techniques in order to create a new model from an existing one or to modify an existing model. Model-Driven Engineering (MDE) belongs to this category: it involves the use of models in development, which entails that at least one User Interface Description Language (UIDL) language must be used and described in terms of the MOF language to enable the metadata to be understood in a standard manner, which is a precondition for any activity to perform automated transformation. Model-to-model (M2M) transformations are also in this case.
8. *Loop on External view*: applies image-processing techniques in order to produce a new external view from an existing one or to modify an existing one.
9. *Loop on Internal view*: applies transcoding to generate UI code in a language other than the original one.

2.2 Historical background: UI views used

UI views have been extensively introduced, motivated, and used so far in order to provide designers and developers with as much guidance as possible during the applications development life cycle. This section provides an historical perspective on UI views.

[20] is one of the pioneer work in this domain, in which the authors remark the importance of taking into account the mental model of the user. As such, for any view (and mainly regarding graphical representations) the users must be able to easily identify and follow the corresponding connections.

FORMSVBT [2] consists of a design environment where two UI views co-exist: an internal view represents the GUI according to LaTeX-like syntax in a dedicated window, and an external view depicts the final UI that can be executed in a second window. The same external view is presented with the hierarchical structure of UI elements in a third window. UI views are *synchronized*: any modification in one view is automatically propagated on the two other views, e.g., moving a widget from one container to another in direct manipulation in the external view, automatically updates the internal view (LaTeX syntax) and the external view without structure.

VISTA [8] synchronizes a conceptual view with an internal view. While the conceptual view is composed of a task model in UAN notation and UAN tables, and presented in two adjacent windows; the internal view is composed of software architecture and code and presented in two other adjacent windows; which totalizes four tiled windows.

TADEUS++ [29] consists of a model-based UI development environment in which a single window contains four related aspects of a conceptual view (i.e., task, domain, and user), and an internal view (i.e., a final UI structure). An external view can be automatically generated at any time in an overlapping window.

TEALLACH [16] maps a conceptual view (consisting of a task, domain, and UI models) onto an external view (consisting of UI elements). Mappings support the developer in propagating the changes of any element in one model in the others, while updating the external view.

IdealXML [24] displays a conceptual view made up of three aspects (i.e., a domain model, a task model, and an abstract UI) in three windows. These three aspects are then represented as trees in order to facilitate editing mappings between elements.

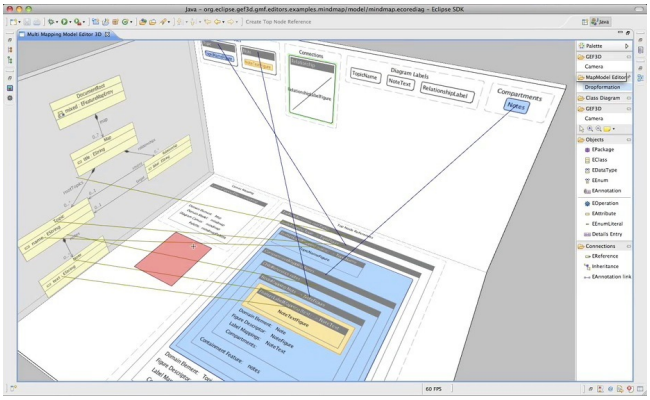

Figure 2. 3D linking between UI views in GEF3D [34]: a conceptual view (UML Class diagram), an external view (a structured UI), and another external view (the final UI).

GEF3D [34] draws lines connecting elements belonging to three

views in 3D: a conceptual view (i.e., a UML class diagram), an external view (i.e., a UI without structure) and a second external view (i.e., a UI with structure). Views are depicted in augmented planes to be assembled together into a volume (Fig. 2).

The aforementioned software and similar others suffer from the following shortcomings with respect to the organization of UI views: (i) *almost inexistent linking*: in many cases, there is no representation of links between views, which may prevent the developer from identifying relationships between views that are subject to synchronization. Coordinated windows are implemented, but without any link representation; (ii) *limited linking representation*: when such a link representation exists (e.g., in Fig. 2), multiple connections may produce a graph that becomes illegible when scalability comes into play, sometimes with a representation that does not facilitate the understanding; (iii) *variation of link representation*: when such a link is represented, many different techniques are used (Table 1); (iv) *no immediate feedback*: due to the lack of link representation, the feedback of updating a content in one view with propagation into another view is often perceived too late by developers; (v) *high cognitive load*: when links are represented between views, the visual density induced by their representation, even on demand, increases the cognitive load for manipulating them.

Dimensions	Links between UI views
1D	Lines in a table (IdealXML [24])
2D	Graphs and trees (FormsVBT [2], Vista [8], Tadeus++ [29], Teallach [16]), drawings (FormsVBT 2])
2½D	Augmented planes (GEF3D [34])
3D	Volumes, such as cubes (GEF3D [34])

Table 1. Links between UI views in terms of dimensions.

In this work, we intend to replace the existing link between UI views by an animated transition in order to address the *mapping problem* in model evolution. The *mapping problem* [11,21,25] addresses the need for expressing links between either different views or different models for the same UI case study. This should not be confused with *model animation*, in which a model is rendered in an animated way, e.g., in order to understand or to simulate it [23].

3. METHODOLOGY FOR ANIMATED TRANSITIONS BETWEEN VIEWS

This section introduces, defines, and explains the various steps required to establish an animated transition between UI views. Fig. 3 shows intermediate steps of an animated transition from a conceptual view to an external view, then from an internal view to an external view. This last transition is discussed in details in the next sub-sections, while the former is similar in principle.

3.1 Step 1. Define the External View

From its definition, the external view is interpreted as the final GUI with the look and feel according to its computing platform. Therefore, the external view will consist of any runnable GUI in any platform that could be expressed in terms of widgets. We choose to implement *UsiView* in Microsoft Expression Studio for the following reasons: (i) it supports XAML, a XML-compliant UIDL that is expressive enough for the external view; (ii) since XAML widgets are vector-based, logical operations such as rotate, enlarge or reduce are easily developed; (iii) basic animated transitions, such as zoom-in or zoom-out, are already built-in with some configuration options; (iv) MS Expression Studio comprises five products: Expression Blend (for building GUIs for Silverlight, Windows, and Surface), Expression Blend SketchFlow (for prototyping these GUIs), Expression Web (for building Web GUIs), Expression Design (for creating graphic assets for the Web or Silverlight, Windows, and Surface), and Expression Encoder (for preparing video assets for the Web or Silverlight, Windows, and Surface); (v) Expression Design will be used to develop the animated transitions and Expression Blend for rendering them. Defining an external view on XAML does not induce any particular restriction and could be based on another UIDL instead without any information loss.

3.2 Step 2. Define the Internal View

From its definition, the internal view is considered as the developer's view in which the UI code or description is manipulated. Today, several XML-compliant User Interface Description Languages (UIDLs), such as XWT (http://wiki.eclipse.org/E4/XWT) [36], XIML (www.ximl.org) [27], or UIML (www.uiml.org) [1], allows describing a GUI. For the purpose of this paper, a subset of UsiXML [33] has been selected based on XWT [36]. Other UIDLs could also be adopted without significant changes.

A GUI is described (Fig. 3b) in a UsiXML [33] document structured into three parts: a *prolog* specifying the header and specification format, a *body* made up of *elements* and *attributes* describing the various GUI widgets, and an *epilog* closing the document. An *element* is considered as the logical document component either beginning with a start-tag and ending up with a matching end tag or being empty. The characters between are called *contents*. Elements can be nested. Elements are refined through *attributes*, a markup construct consisting of a pair (*variable name*, *value*) that exists within a start-tag or an empty tag. UsiXML [33] elements determine one or many instances of a widget of a certain type. For example, the element <textfield …/> </textfield> (Fig. 3b) determines an instance of an edit field in a form (Fig. 3c). Attributes specify one or many properties of an existing widget. For example, the attribute name="Firstname" (Fig. 3b) is used to identify the edit field and to have the prompt "Please enter your first name" in the field (Fig. 3c). Element nesting determines the structure of existing widgets and their layout. For example, the hierarchy of group in Fig. 3b is rendered into tabbed dialog box in Fig. 3c.

3.3 Step 3. Define the Mapping between Views

In order to define a mapping between UI views, taking as example from the internal view to the external view, a correspondence must be established and maintained between elements belonging to the internal view and elements belonging to the external view. This mapping is structured according to the following format:

Mapping M: series of pairs (variable name, value) ↦ *set of instructions on the widgets of the external view.*

A mapping relates a series of pairs (variable name, value) belonging to the internal view to a set of instructions to be executed on the widgets belonging to the external view. These instructions are structured according to an Extended Backus-Naur Form (EBNF) grammar. Four mapping types could be distinguished depending on their input/output.

M1*:* element specifying a widget type *T* ↦ create an instance of widget type *T* with associated properties. The associated properties are themselves regulated by the following sets of rewriting rules: rules for selecting widgets or rules for layout production (that is itself decomposed into *location, sizing*, and *arrangement*). For example, the pair (hint, "form") parsed from the UsiXML [33] document in Fig. 3b is mapped onto the

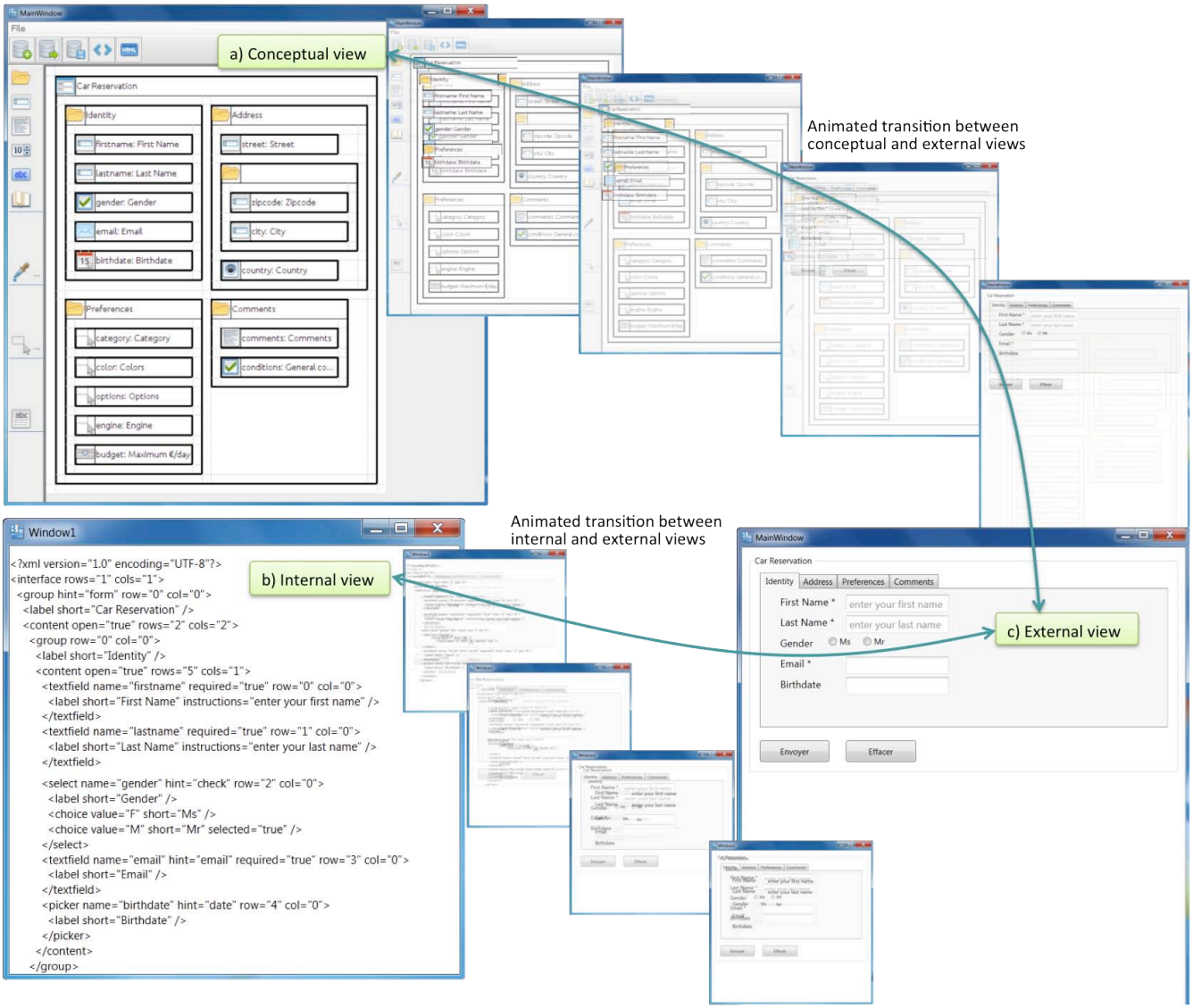

Figure 3. Starting point, intermediate steps, and ending points of animated transitions between user interface views.

following set of instructions in order to automatically create a form with submit and cancel buttons:

Create (w, window),
Create (s, pushbutton, w), Set (s, label, "Submit")
Create (t, pushbutton, w), Set (t, label, "Cancel")
Set (s, position, bottom-left), Set (t, position, bottom-left-row)

M2: attribute specifying a property P of a widget of type T ➥ assign values to properties of a widget instance of type T. For example, the pairs (rows, "0"), (cols, "0") in Fig. 3b based on their parents (rows, "1"), (cols, "1") together define an offset of (1,1) for displaying the form, thus resulting into the following pair of instruction Set (w, X_pos, "1"), Set (w, Y_pos, "1").

M3: attribute specifying a property P of a widget instance of type T ➥ assign value to properties of an existing widget instance. For example, the pair (label_short, "Car reservation") (Fig. 3b) leads to the instruction Set (w, label_short, "Car reservation") in order to create the label of the corresponding tab in Fig. 3c.

M4: compute height/length of a widget type T governed by element nesting ➥ assign value(s) to properties of an existing widget instance. For example, the functions $h = \Sigma\, h_i + 2$ spaces, respectively $l = \Sigma\, l_i$ + (i-1) spaces, determine the total height, respectively the total length, of the form depending on the individual heights (h_i) and lengths (l_i) of contained widgets. This will lead to the two instructions: Set (w, length, l), Set (w, height, h).

3.4 Step 4. Derive the Transition from the Mapping Definition

A *transition* is hereby defined as the logical way to transform the input of a mapping into its output depending on their respective data type (e.g., text, color, shape). A transition is therefore encoded in *UsiView* by an identifier, a name, a list of synonyms, a description, a transition type (e.g., text-to-text, text-to-color, text-to-shape), and a transition cardinality that is defined as follows:

- *One to one*: one element belonging to the initial view (the internal view in our running example) is mapped onto one element belonging to the final view (the external view in our running example). For example, assign a label to a widget (*M3*).
- *One to many*: one element belonging to the initial view is mapped onto many elements belonging to the final view. For example, create an instance of a widget type (*M1*) or assign the same foreground color to a set of widget instances (*M2*).
- *Many to one*: many elements belonging to the initial view are mapped onto one element belonging to the final view. For

example, the foreground color of one widget in a web page is determined by considering both its HTML code and CSS.

- *Many to many*: many elements belonging to the initial view are mapped onto many elements belonging to the final view. For example, HTML and CSS together determine the border color of several widgets included in a container.

In order to determine the transition type, coding schemes [22] have been gathered and sorted by decreasing level of precision. Fig. 4 ranks coding schemes by decreasing order of precision respectively for a quantitative variable, for an ordinal variable, and for a nominal variable. The first column of Fig. 4 is read as follows: in order to represent a quantitative variable, the position is the most precise variable, followed by the length, the angle, etc.

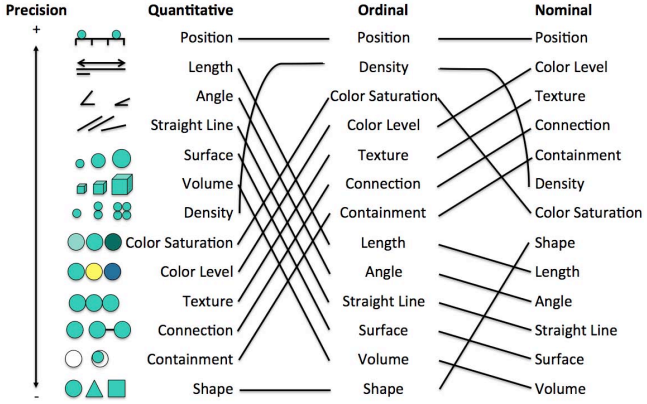

Figure 4. Coding schemes by decreasing level of precision.

Based on this classification (Fig. 4), Table 2 summarizes the transition types from an internal view to an external view:

1. A *text-to-text* transition type denotes a transition where a textual element of the initial view (e.g., some UsiXML text in the internal view) is mapped onto a textual element of the final view (e.g., a label in the external view).
2. A *text-to-position* transition type denotes a transition where a textual element of the initial view (e.g., some UsiXML text in the internal view) should be mapped onto a position of an object (e.g., the location of a widget).
3. A *text-to-length* transition type denotes a transition where a textual element of the initial view (here, some UsiXML text in the internal view) should be mapped onto the length of an object (e.g., the size of a widget).
4. A *text-to-color-saturation* transition type denotes a transition where a textual element of the initial view (e.g., some UsiXML text in the internal view) should be mapped onto the saturation of a color of an object (e.g., the foreground color of a widget). Note that color level and texture are handled similarly by playing with the RGB code of any color.
5. A *text-to-shape* transition type denotes a transition where a textual element of the initial view (e.g., some UsiXML text in the internal view) should be mapped onto a shape of an object (e.g., a rectangle for a text field, a circle for a radio button). A variant of this transition is the *text-to-symbol*, where the shape is replaced by a special symbol (e.g., * for a mandatory field).

For the moment, *UsiView* does not support four of the transition types mentioned above (i.e., text-to-surface, text-to-volume, text-to-density, and text-to-connection) because situations involving such transition types have not yet been met and because they require more advanced development based on animation techniques –such as shape, surface, or volume deformation and morphing– found in information visualization [27] or computer graphics [30,32]. SMIL (Synchronized Multimedia Integration Language), for instance, as a markup language capable of describing multimedia presentations, can be applied in this sense to support the implementation the animations across different views [35].

3.5 Step 5. Identify Animation Technique to Produce the Animated Transition

This section discusses which animation technique could be used in order to animate a transition according to its transition type.

Text-to-text. This transition typically involves texts of different fonts, sizes, and styles. Some text animation techniques, like text mapping [10] and pixel-based approach [14], address this issue. We chose a more simple approach that was made possible thanks to the vectorial rendering of XAML:

- ¬ When the initial text is *identical* to the final text, the text is simply moved from its initial to its final coordinates computed according to the objects position in their respective views.
- ¬ When the initial text is *different* from the final text, the initial text is again moved to its final coordinates and then replaced by the final one using a fade in/fade out visual effect [12,30].
- ¬ When the initial text and the final text are different in fonts, sizes or styles, the initial text is again moved to its final coordinates and aligned in font, size, and style by a box in/out visual effect [12], whether the texts are identical or different.

These rules adhere to the main principle of moving the initial text first to its final location to see its impact, then to transform it depending on the conditions. This is applicable for any text: label, push button label, edit field label, message inside an edit field, a tab label, window name, group box label, etc.

Transition	Internal view	External view
Text-to-text	<label_short="Birthdate">	Birthdate :
Text-to-position	<textfield … col="4"…>	4 Birthdate :
Text-to-length	<textfield … length="20".>	20 Birthdate :
Text-to-color-saturation	<textfield. fgColor="red".>	Birthdate :
Text-to-color-texture	<textfield … bgTexture="checkerboard"… >	Birthdate :
Text-to-shape	<textfield name="Birthdate">	Birthdate :
Text-to-symbol	<textfield … required="yes"… >	Birthdate * :

Table 2. Transition types from internal view to external view.

Text-to-color. In order to render a color (e.g., based on saturation level or texture), the metaphor used is the painter palette according to these rules:

- ¬ The initial text representing the color (e.g., the foreground color of a widget, the background color of a tab, a texture color of an area) is first colored according to the target color using Red-Green-Blue (RGB) transformations.
- ¬ The colored text is then replaced by a colored box by a "Box in/out" visual effect [12] and the texture is applied if needed.
- ¬ The new colored box is then moved from its initial coordinates to the final coordinates using a fade in/out effect whose direction is governed by the arrow from the source to target.

Text-to-shape. Gliimpse [14] renders this transition type by linearly interpolating the bounding boxes of the initial and final

objects and by alpha-blending them. [12] uses a "barn door close" visual effect to close a transient screen or a current scene and a "barn door open" visual effect to open a transient screen, to initiate a new step and to open a new window. In order to simplify the transition, the initial text is first replaced by its bounding box on the line where it starts (if it spans over several lines, the bounding box remains on the first line) using a "barn door close" visual effect and then replaced with the final shape using a "barn door open" visual effect. The new element is then moved to its final coordinates. This transition is used for any new shape or symbol (the two last lines of Table 2). Shape-to-shape transitions are handled similarly except that there is no bounding box created, but simply the initial shape is replaced by its respective final shape.

Disappearing elements. Any element of the initial view that does not map to any element of the final view simply disappears using a fade out visual effect. For example, in the animated transition from Fig. 3a to Fig. 3c the shape-to-shape transitions are used to map the model elements of the conceptual view onto their corresponding widgets for the tab displayed; all the other elements simply vanish progressively since they have no counterpart in the external view at hand. Selecting another tab in the external view activate the corresponding model elements in the conceptual view.

3.6 Executing the Animated Transition

Fig. 3a reproduces a screenshot of the UsiXML-based simplified GUI builder developed for the purpose of this paper, in which the developer could drag and drop model elements from the palette to the working area, thus producing the UsiXML corresponding code. This code could also be manually typed and synchronized with the model. At any time, the animated transition is triggered by pressing the <Ctrl> key once for one-way and twice for two-way. Pressing simultaneously the "+" or "-" key increases or decreases the animation speed, in order to address the lag problem [4,10]. In this context the user manually defines the timing for the animation, however automated or hybrid approaches could also be adopted. For example considering the approach of [13] in which animations that start and finish in a slower speed than the intermediary one, are better perceived by the end users.

4. RELATED WORK

4.1 Animation

"Smooth interactive animation is particularly important because it can shift a user's task from cognitive to perceptual activity, freeing cognitive processing capacity for application tasks." [30] summarizes the benefits of using animation in GUIs. *Animation* [3, 5, 30] has been widely used as a general technique for supporting end users in understanding various phenomena: the evolution of a dynamic process, a chronological sequence of events, complex graphics and statistics [18], relations between elements such as spatial connections [5], for organizing diagrams [6], for improving decision making [15] and for searching information in 3D tree-maps [27]. Small animated icons are proved to convey functionality better than static icons of the same size [19].

4.2 Animated Transitions

Animated transitions [4,6,10,12,14,18,19,28,32] in interactive systems are aimed at conveying to the end user a transition between states, views or scenes, e.g., to foster a smooth progression between two scenes, menus, widgets [4] or images [19]. Animated transitions improve feedback on users' actions, notify display changes, and improve situation awareness in a distributed environment.

Animated transitions are subject to a series of potential *shortcomings* [4,5,10,14]: they may require more cognitive workload than static images, they attract the end user's attention first, they may cause user distraction, their duration always induce some lag [4,30], their execution requires additional processing capabilities, the animated objects should not exceed a certain threshold. To minimize lag, an animated transition should be fast, but not too fast, otherwise the end user may completely overlook the animated transition [10]. Variable speeds can be also considered to play the transitions [13].

Animated transitions may induce a significantly *positive impact* on understanding display changes, whether it is for notifying value changes in GUI widgets [4], for updated contents in a web page [31], such as web navigation [17] or for evolving data in a dynamic display. Different techniques support end users in perceiving and understanding screen changes, mainly based on animation between states [19], perhaps supplemented by sound [28].

Mnemonic rendering [7] consists of an image-based technique that buffers all changes of a fast-changing dynamic display and restitutes these changes under the end user's control via a memory jog.

DiffIE highlights web page contents that have been updated since last visit [31]. A positive value has been demonstrated on how people interact with the web page and understand their contents, in particular dynamic ones.

Phosphor widgets [4] rely on afterglow visual effect in order to leave some visual reminiscence of changes of widgets values (e.g., the value change of a slider, the check/uncheck of a check box, a new selection in a radio box).

Fialho & Schwabe [5] enrich the user experience of web applications by applying a *Rhetoric Structure Theory* (RST) as a way to set the effects presented by animation, as well their sequence and duration. In order to capture the dynamic aspects of a widget, an Abstract Widgets Ontology (AWO) was extended to include the following classes: *Transition* (for representing a state change), *RhetoricalStructure* (for animating a transition or in response to an event), and *Decoration* (for animating a widget change). The Decoration class is further refined into the following elements: InsertElement (for introducing a new element), RemoveElement (for removing an element from the destination state), MatchElements (for matching the parameters of an element in the current state and another in the destination state), TradeElements (for performing a transformation of an element in the current state into another in the destination state), and EmphasizeElement (for highlighting an action).

Differentiated transitions [28] are animated transitions that support explaining a process over time in a way that is reflected in the visual effect. For instance, the transfer time, the network bandwidth, and the file size are explicitly represented in an animated transition depicting a file transfer.

RST, respectively Mnemonic rendering, force end users to wait for, respectively to replay, the display changes, thus inducing some lag [30]. DiffIE does not induce such a drawback (since the highlighting is almost instantaneous), nor Phosphor widgets (since the afterglow effect does not stop user in their tasks). Differentiated transitions actually animate the task while being executed [28], thus not representing any hindrance for achieving the end user's goals.

4.3 Animated Transitions between Views

Most aforementioned techniques apply animated transition only on one UI view at a time. For instance, Phosphor Widgets and Differentiated transitions are applied on the external view only.

RST [17] is applying animated transitions on two views separately, but not across views: an abstract UI description of a web page as a conceptual view and an external view consisting of the web page.

The closest work to *UsiView* is Gliimpse [14] in which an animated transition is ensured between an internal view (e.g., HTML) and an external view (e.g., a web page), but no explicit link with the conceptual view. In Gliimpse [14], a web form can also be rendered from its HTML code. In *UsiView*, an XML representation (i.e., UsiXML) is used instead of HTML in order to support independence between the internal view and the external view since this XML representation could be obtained from other languages (e.g., HTML, XForms, XWT, Java classes) via XSLT transformations. In addition, a mapping mechanism has been introduced in order to logically define how the animated transition will be produced. This definition could be modified or could use a different set of rules for choosing which animated technique should be selected at run-time.

5. CONCLUSION AND PERSPECTIVES

This paper presented *UsiView*, a technique supporting animated transitions between three UI views: conceptual, internal, and external. The potential benefits of this approach are: reducing the cognitive load induced by task switching and view switching, improving the understanding of the impact of one view on another, reducing the task completion time for simple view editing actions.

Animated transitions could be used in many different situations: from one view to another, between coordinated views, when several levels of abstractions (e.g., in CRF [9]) co-exist, when transformations are applied between them.

Future work will consider extending the transition types for the currently supported animated transitions, but also investigate to what extent different views, possibly of the same type, could be derived from the same one. For instance, different internal views could be offered depending on the user level of experience, different external views from the same internal view could be provided on-demand. As the software development life cycle progresses, it is likely that some views will evolve. Therefore, it is also worth to investigate how to ensure an animated transition from one view over time in order to provide a visual feedback on view evolution, similarly to Mnemonic Rendering [7]. For example, an animated transition between conceptual views, internal views or external views (defined as loops on views in Section 2) over time could support the design history. Similarly, reverse engineering techniques could be investigated to recover an internal or conceptual view from an external view while explaining how this reverse engineering technique has produced which result. For example, when a GUI screen shot is available (an external view), a conceptual view could be derived from it.

Another way of investigation consists of exploring an animated transition of some elements at once (e.g., to preview only a group of elements, not all of them) or only one element at a time (e.g., to preview the effect of one tag in the internal view on the widgets of the external view), as opposed to all elements at once as it is today.

6. REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E., UIML: An Appliance-Independent XML User Interface Language, in Proc. Of the 8th World Wide Web Conf. WWW'8 (Toronto, May 11-14, 1999), Computer Networks, 1999.

[2] Avrahami, G., Brooks, K.P. and Brown, M.H. 1989. A Two-view Approach to Constructing User Interfaces. In *Proc. of 16st Annual Conf. on Computer Graphics and Interactive Techniques* (Boston, 31 July-4 August 1989). SIGGRAPH'-89. *Computer Graphics 23*, 3 (July 1989), 137-146.

[3] Baecker, R. and Small, I. 1990. Animation at the interface. In *The Art of Human-Computer Interface Design*, B. Laurel, Ed. Addison-Wesley, New York, 251-267.

[4] Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G. 2006. Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects. In *Proc. of ACM Symposium on User Interface Software Technology* (Montreux, October 15-18, 2006). UIST'2006. ACM Press, New York, 169-178.

[5] Bederson, B.B. and Boltman, A. 1999. Does Animation Help Users Build Mental Maps of Spatial Information? In *Proc. of IEEE Symposium on Information Visualization*. InfoVis'99. IEEE Computer Society Press, Los Alamitos, 28-35.

[6] Bladh, T., Carr, D.A., and Kljun, M. 2005. The Effect of Animated Transitions on User Navigation in 3D Tree-Maps. In *Proc. of the 9th Int. Conf. on Information Visualization*. InfoVis'2005. IEEE Computer Society, Los Alamitos, 297-305.

[7] Bezerianos, A., Dragicevic, P., and Balakrishnan, R. 2006. Mnemonic Rendering: An Image-Based Approach for Exposing Hidden Changes in Dynamic Displays. In *Proc. of ACM Symposium on User Interface Software Technology*. UIST'2006. ACM Press, New York, NY, 159-168.

[8] Brown, J., Graham, T.C.N., and Wright, T.N. 1998. The Vista Environment for the Coevolutionary Design of User Interfaces. In *Proc. of ACM Conf. on Human Aspects in Computing Systems*. CHI'1998. ACM Press, New York, 376-383.

[9] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. 2003. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers 15*, 3 (June 2003), 289-308.

[10] Chevalier, F., Dragicevic, P., Bezerianos, A., and Fekete, J.-D. 2010. Using Text Animated Transitions to Support Navigation in Document Histories. In *Proc. of ACM Conf. on Human Aspects in Computing Systems* (Atlanta, April 10-15, 2010). CHI'2010. ACM Press, New York, 683-692.

[11] Clerckx, T., Luyten, K., and Coninx, K. 2004. The mapping problem back and forth: customizing dynamic models while preserving consistency. In *Proc. of 3rd Conf. on Task models and diagrams*. TAMODIA'2004. ACM Press, NY, 33-42.

[12] Dessart, Ch.-E., Motti, V. G., and Vanderdonckt, J. 2011. Showing User Interface Adaptivity by Animated Transitions. In *Proc. of 3rd ACM Symposium on Engineering Interactive Computing Systems*. EICS'2011. ACM Press, NY, 95-104.

[13] Dragicevic, P., Bezerianos, A., Javed, W., Elmqvist, N., and Fekete, J.-D.. 2011. Temporal distortion for animated transitions. In *Proceedings of the 2011 annual conference on Human factors in computing systems* (CHI '11). ACM, New York, NY, USA, 2009-2018. DOI=10.1145/1978942.1979233 http://doi.acm.org/10.1145/1978942.1979233

[14] Dragicevic, P., Huot, S., and Chevalier, F. 2011. Gliimpse: Animating from Markup Code to Rendered Documents and Vice Versa. In *Proc. of 24th ACM Symposium on User Interface Software and Technology*. UIST'2011. ACM Press, New York, 257-262.

[15] Gonzalez, C. 1996. Does animation in user interfaces improve decision making? In *Proc. of ACM Conf. on Human Aspects in Computing Systems* (Vancouver, April 13-18, 1996). CHI'1996. ACM Press, New York (1996), pp. 27-34.

[16] Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J.B., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro da Silva, P. 2011. Teallach: a model-based user interface development environment for object databases. *Interacting with Computers 14*, 1, 31-68.

[17] Fialho, A.T.S. and Schwabe, D. 2007. Enriching Hypermedia Application Interfaces. In *Proc. of 7th Int. Conf. on Web Engineering* (Como, July 16-20, 2007). ICWE'2007. LNCS, Vol. 4607. Springer, Berlin, 188-193.

[18] Heer, J. and Robertson, G. 2007. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (Nov. 2007), 1240-1247.

[19] Huhtala, J., Sarjanoja, A.-H., Mäntyjärvi, J., Isomursu, M. and Häkkilä, J. 2010. Animated UI transitions and perception of time: a user study on animated effects on a mobile screen. In *Proc. of ACM Conf. on Human Aspects in Computing Systems*. CHI'2010. ACM Press, New York, 1339-1342.

[20] Kazuo Misue, Peter Eades, Wei Lai, Kozo Sugiyama. 1995. Layout Adjustment and the Mental Map. J. Vis. Lang. Comput. 6(2): 183-210.

[21] Limbourg, Q. and Vanderdonckt, J. 2004. Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design* (Prague, November 15-16, 2004). Tamodia'2004. ACM Press, New York, 155-163.

[22] Mackinlay, J. 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics 5*, 2 (April 1986), 110-141.

[23] Mirlacher, T. 2011. Modeling Animations for Dependable Interactive Applications. In *Proc. of 3rd ACM Symposium on Engineering Interactive Computing Systems* (Pisa, June 13-16, 2011). EICS'2011. ACM Press, New York, 319-322.

[24] Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M.D., and Limbourg, Q. 2005. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In *Proc. of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems*. DSV-IS'2005. LNCS, Vol. 3941, Springer, Berlin, 161-172.

[25] Puerta, A.R. and Eisenstein, J. 1999. Towards a General Computational Framework for Model-Based Interface Development Systems. In *Proc. of ACM COnf. on Intelligent User Interfaces*. IUI'1999. ACM Press, New York, 171-178.

[26] Puerta, A. R. and Eisenstein, J. 2002. XIML: a common representation for interaction data. In Proceedings of the 7th international conference on Intelligent user interfaces (IUI '02). ACM, New York, NY, USA, 214-215. DOI=10.1145/502716.502763 http://doi.acm.org/10.1145/502716.502763

[27] Robertson, G.G., Mackinlay, J.D., and Card, S.K. 1991. Information visualization using 3D interactive animation. In *Proc. of the ACM Conf. on Human factors in computing systems*. CHI'1991. ACM Press, New York, 461-462.

[28] Schlienger, C., Dragicevic, P., Ollagnon, C., and Chatty, S. 2006. Les transitions visuelles différenciées : principes et applications. In *Proc. of 18th Int. Conf. of the Association Francophone d'IHM* (Montréal, April 18-21, 2006). IHM'2006. ACM Int. Series, Vol. 133. ACM Press, 59-66.

[29] Stary, Ch. 2000. TADEUS: seamless development of task-based and user-oriented interfaces. *IEEE Transactions on Systems, Man, and Cybernetics, Part A 30*, 5, 509-525.

[30] Stasko, J. 1993. Animation in User Interfaces: Principles and Techniques. In *Proc. of User Interface Software '93*, 81-101.

[31] Teevan, J., Dumais, S.T., Liebling, D.J., and Hughes, R. 2010. A Longitudinal Study of How Highlighting Web Content Change Affects People's Web Interactions. In *Proc. of ACM Conf. on Human Aspects in Computing Systems*. CHI'2010. ACM Press, New York, 1353-1356.

[32] Thomas, B.H. and Calder, P. 2001. Applying Cartoon Animation Techniques to Graphical User Interfaces. *ACM Trans. on Computer-Human Interaction 8*, 3 (Sept. 2001), 198-222.

[33] Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).

[34] von Pilgrim, J. and Duske, K. 2008. GEF3D - a Framework for Two-, Two-and-a-Half-, and Three-Dimensional Graphical Editors. In *Proc. of the 4th ACM Symposium on Software Visualization*. SoftVis'08. ACM Press, New York, 95-104.

[35] W3C. SMIL Animation, W3C Recommendation 04-September-2001http://www.w3.org/TR/smil-animation/

[36] XWT (XML Window Toolkit). Available at: www.xwt.org.